Addressing False Data Injection Attacks in IoT Systems with a Domain Specific Language within an Industrial Context

Mathieu Briland

THÈSE DE DOCTORAT

DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

PRÉPARÉE À L'UNIVERSITÉ DE FRANCHE-COMTÉ

École doctorale n°37

Sciences Pour l'Ingénieur et Microtechniques

Doctorat d'Informatique

par

MATHIEU BRILAND

Addressing False Data Injection Attacks in IoT Systems with a Domain Specific Language within an Industrial Context

Adresser les attaques par injection de fausses données dans les systèmes IoT à l'aide d'un langage dédié dans un contexte industriel

Thèse présentée et soutenue à Besançon, le 28 mai 2021

Composition du Jury :

BOUQUET FABRICE	Professeur à l'Université de Franche-Comté	Directeur de thèse
GUIOT YOHANN	RSSI du groupe Flowbird	Examinateur
LE TRAON YVES	Professeur à l'Université du Luxembourg	Rapporteur
LEGEARD BRUNO	Professeur à l'Université de Franche-Comté	Président du jury
PARISSIS IOANNIS	Professeur à l'Université de Grenoble Alpes	Rapporteur

N° | X | X | X

ACKNOWLEDGEMENTS

Tout d'abord, je tiens à exprimer mes sincères remerciements à mon directeur de thèse Fabrice Bouquet, pour son soutien tout au long de cette thèse, ses innombrables conseils et heures passées en réunion, qui a permis l'aboutissement de ce travail. Merci également à tous les membres des équipes GeLeaD et SARCoS, pour m'avoir fait découvrir le plaisir de la recherche au sein d'une grande équipe.

Je remercie Yves Le Traon et Ioanis Parissis, qui ont accepté de relire cette thèse et d'en être les rapporteurs.

Je remercie Bruno Legeard et Yohann Guiot d'avoir accepté d'être examinateurs pendant la soutenance de cette thèse.

Merci à tous les collaborateurs de Flowbird qui ont rendu cette thèse possible, en particulier Luc Porchon qui m'a encadré, conseillé et débloqué dans cette grande entreprise.

Un grand merci a Jérémy Bourdé et Thomas Cantenot de leur aide lors des longues sessions de relecture que nécessite les travaux d'une thèse.

Également, merci à tous les membres de l'équipe ArchiPEL, qui m'ont permis de m'intégrer rapidement dans l'entreprise malgré leur désamour des visiteurs.

Merci à tous mes proches qui m'ont soutenu, je pense notamment à mes parents qui ont toujours été là derrière moi, et tous les membres et amis de Pixel bisontin, merci pour votre soutien.

And last but not least, merci Hélène de m'avoir donné le courage d'aller au bout de ce travail, merci pour tout.

À tous ceux que j'ai malheureusement oubliés, merci.

Contents

I Introduction and Context 1

1 Introduction 3

1.1 Motivation . . . 3

1.2 Internet of Things Cybersecurity . . . 5

1.3 Research Questions . . . 6

1.4 Contribution of the Thesis . . . 7

1.4.1 Approach Definition to Address the FDIA Challenge . . . 8

1.4.2 Sensors and Data Classification . . . 8

1.4.3 Domain-Specific Language . . . 9

1.4.4 Prototype Tool . . . 9

1.4.5 Experimentation . . . 9

1.4.6 Research Project . . . 10

1.4.6.1 SARCoS Project . . . 10

1.4.6.2 GeLeaD Project . . . 10

1.5 Organisation of the Dissertation . . . 11

2 Flowbird Information System Security 13

2.1 Information System Presentation . . . 14

2.1.1 Parking Meter . . . 14

2.1.2 Network . . . 16

2.1.3 Data Centre . . . 16

2.1.4 Third Parties . . . 17

2.2 Risk Analysis . . . 17

2.2.1 Parking Meter . . . 17

2.2.2 Network . . . 19

2.2.3 Data Centre . . . 19
2.3 Internet of Things Analogy . . . 20
2.3.1 Perception Layer . . . 21
2.3.2 Network Layer . . . 21
2.3.3 Middleware Layer . . . 22
2.3.4 Application Layer . . . 22
2.3.5 Business Layer . . . 22
2.4 Conclusion . . . 22

3 State of the Art 25
3.1 Internet of Things Security . . . 26
3.1.1 Perception Layer . . . 27
3.1.2 Network Layer . . . 28
3.1.3 Middleware Layer . . . 28
3.1.4 Application Layer . . . 29
3.1.5 Business Layer . . . 30
3.2 False Data Injection Attack . . . 30
3.2.1 FDIA Definition . . . 32
3.2.2 FDIA Research . . . 34
3.2.3 Related Works . . . 35
3.3 FDIA Testing . . . 37
3.3.1 Model-Based Testing . . . 37
3.3.2 Data Generation . . . 39
3.3.2.1 Generation Techniques . . . 39
3.3.2.2 Generated Data . . . 42
3.4 Conclusion . . . 43

II Contribution 45

4 Data Analysis 47
4.1 Sensor Classification . . . 47

4.2 Data Classification 50
4.3 Conclusion 52

5 Domain Specific Language for FDIA Design 53

5.1 Decision 54
5.2 Domain Analysis 56
5.2.1 Scenario 57
5.2.2 Scenario Properties 57
5.2.3 Scenario Actions 58
5.2.4 Selection Criteria 58
5.2.5 Alteration Criteria 59
5.2.6 Time Frame 59
5.3 Design 59
5.3.1 Scenario 60
5.3.2 Scenario Actions 61
5.3.3 Selection Criteria 62
5.3.4 Alteration Criteria 63
5.3.5 Time Frame 64
5.3.6 Grammar Utility 64
5.4 Implementation 65
5.4.1 Language Implementation 65
5.4.2 Language Interpretation 67
5.5 Conclusion 67

6 FDIA Framework 69

6.1 Workflow of our Approach 69
6.2 DSL Usage 71
6.2.1 Data Management 71
6.2.2 Validation of Alteration Primitive 73
6.2.3 Coverage of the Knowledge 74
6.2.3.1 Expressiveness and Domain Coverage of Primitive 75

6.2.3.2 Coverage of the Data . . . 77
6.2.3.3 Demonstration of the Interest . . . 78
6.3 Conclusion . . . 79

III Experimentation and Conclusion 81

7 Experimentation 83
7.1 FDIA on Industrial Context . . . 84
7.1.1 Simple FDIA - Sensor Failure . . . 84
7.1.2 Mutli-Sensor FDIA - Attack on Particles Sensors . . . 85
7.1.3 Private Companies . . . 87
7.1.4 Municipality . . . 88
7.1.5 Law Enforcement . . . 88
7.1.6 Devices User . . . 89
7.1.7 Other Services . . . 89
7.1.8 Devices Manufacturers and Operators . . . 89
7.2 Performance and Efficiency . . . 90
7.3 FDIA Experimentation Synthesis . . . 92
7.4 Machine Learning Experimentation . . . 93
7.4.1 Data Synthesis . . . 93
7.4.2 Detection of Attacks . . . 95
7.4.3 Discussion . . . 97
7.5 Conclusion . . . 98

8 Conclusion 101
8.1 Summary . . . 101
8.2 Future Works . . . 103
8.2.1 Overall DSL Improvement . . . 103
8.2.2 Testing Phase . . . 104
8.2.3 Machine Learning Usage . . . 105

Bibliography 107

List of Figures 115

List of Tables 118

List of Definitions 119

Acronyms 123

IV Appendix 125

A ANTLR grammar definition 127

I

Introduction and Context

1

INTRODUCTION

Contents

1.1 Motivation . . . 3
1.2 Internet of Things Cybersecurity . . . 5
1.3 Research Questions . . . 6
1.4 Contribution of the Thesis . . . 7
1.4.1 Approach Definition to Address the FDIA Challenge . . . 8
1.4.2 Sensors and Data Classification . . . 8
1.4.3 Domain-Specific Language . . . 9
1.4.4 Prototype Tool . . . 9
1.4.5 Experimentation . . . 9
1.4.6 Research Project . . . 10
1.5 Organisation of the Dissertation . . . 11

This thesis was carried out as part of a CIFRE[1] agreement. It was conducted within the Flowbird[2] company and the Université de Bourgogne Franche-Comté in the Département Informatique des Systèmes Complexes (DISC) of the Femto-ST research institute. The work presented here was carried out from September 2017 to December 2020 in Besançon, France.

1.1/ MOTIVATION

The primary motivation for this work comes from the core business of the Flowbird company. Flowbird is the world leader in on-street parking solutions. The company has been producing internet connected parking control devices for 20 years. These parking meters are present in the streets of thousands of cities around the world, and are therefore subject to many threats from their environment, both natural and malicious. The parking meters are connected to the internet, have embedded computing power, are subject to

[1] Conventions Industrielles de Formation par la REcherche (Industrial Research Training Agreements)
[2] http://www.flowbird.group

autonomy issues and to physical threats from their environment. They share the same properties as Internet of Things (IoT) devices. In this work, we therefore consider the devices produced by Flowbird as part of the IoT.

In the last 20 years, technologies have evolved a lot, as have parking meters. This is the main motivation of this work, as the company has recently chosen to make its parking meters evolve technologically. The first generations of devices, being forerunners in their fields, were developed on ad hoc technological bases. They were therefore proprietary technologies that were not thoroughly documented, even to an extent within the company. This had the effect of protecting the devices through a limited access to the knowledge of their inner workings, a kind of opacity. This design is known as *security by obscurity* [Mercuri and Neumann, 2003] and often lead to a false sense of security. Even if attempting to attack such kinds of old and obscure systems could prove time-consuming, and therefore not worthwhile to an unmotivated attacker, the same systems often hide potential vulnerabilities, hence becoming interesting to a more determined attacker. For the same reasons, it is also time-consuming as well as expensive for the company to maintain these systems.

In response, the company chose to create a new and more efficient parking meter ecosystem based on recent, open and widespread technology used in IoT, such as the Android operating system or the MQTT[3] protocol. The use of this kind of technology allows switching to the principle of *security by design* [Santos et al., 2017]. It is the idea of integrating security at the very basis of any project, especially on the architecture level, to make the resulting system robust. As a result, the worst-case scenario is usually considered the default situation, i.e., where the attacker has access to the source code. It then becomes interesting to make the source code open, so as not to rely on the false sense of security provided by secrecy. The drawback of open source codes is that, when a new security flaw is discovered (known as 0-day), the systems built with them are exposed to multiple attackers. If the systems are not kept up to date, even lesser skilled attackers, often referred to as script kiddies, could compromise them, using tools easily found on the internet. Fortunately, developers of open-source software are very responsive, and can be warned of a potential flaw by the large, and often benevolent, community, which means that security patches are published quickly. Nevertheless, many attacks on IoT systems have not yet been properly studied (see State of the Art section below) and need to be addressed, specifically data integrity issues. Indeed, the trust and reliability of collected and processed data is of importance in a commercial activity like Flowbird's.

Thus, in regard to the potential threats to these new technologies, the vulnerable location of Flowbird devices in the streets, and the criticality of data integrity in this business sector, we have chosen to address integrity issues, and in particular the False Data Injection

[3]Message Queuing Telemetry Transport

Attack (FDIA), in the context of Flowbird devices, but also for IoT systems in general, as they are potentially vulnerable to the same attack.

1.2/ INTERNET OF THINGS CYBERSECURITY

In recent years, the IoT has gained great popularity: billions of objects connected to each other or to the internet surround us, both in the industrial and private spheres. The International Telecommunication Union (ITU) defines the IoT as:

> **Definition 1: Internet of Things**
>
> "A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies." [ITU, 2012]

The IoT is a network of networks, it is composed of many technologies and many heterogeneous devices (things). By 2025, [Analytics, 2019] estimate the number of IoT devices to 38.6 billion, and 50 billion in 2030. ITU defines the things as:

> **Definition 2: Thing**
>
> "With regard to the Internet of things, this is an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks." [ITU, 2012]

The rapid increase and mass adoption of IoT in our society, coupled with physical vulnerability and lack of standardisation, has led malicious attackers to take an interest in IoT systems. In [NETSCOUT, 2018] they show that IoT devices are attacked on average within five minutes after being connected to the internet and targeted by specific exploits in the first 24 hours. Our society is using more and more connected objects every year, and even depends on them in some areas. This rapid increase, the youth of IoT, as well as its very nature of being ubiquitous, combined with many protocols and heterogeneous hardware, has led many IoT manufacturers to ignore the security challenges posed by IoT in its early years. As adoption increases, so does the risk of security threats and today even with the steady improvement in terms of security, many challenges still need to be addressed, especially for data integrity, data privacy and data availability. Data are so important in IoT because they are the very foundation of IoT. IoT aims to connect the physical world and the virtual world, and this connection is made through data, by a constant bidirectional communication between the IoT devices and their back offices. On some specific IoT area such as smart-health, the data collected by the things are confidential and critical, they should not be accessed by anyone without authorisation or consent, this is the privacy. Also, data collected by IoT devices are then processed, either

to perform action or for analysis purpose, which makes the availability and integrity of the data essential. Indeed, if these two data security requirements are not met, the entire IoT system and related services will stop working, either because of a lack of data or because of operating errors. The data availability and data integrity are then fundamental to the proper functioning of any IoT system, while data privacy is domain dependant.

We choose in this thesis to work on data integrity, and especially on a type of attack that aims to disrupt the data quality by injecting fake fabricated data on IoT systems. This type of attack is known as False Data Injection Attack (FDIA). It usually has two objectives, the first being to modify the data to trigger automatic and specific actions, and the second being to break the trust placed by users in the system.

1.3/ Research Questions

The main challenge studied in this thesis is "How to verify and assess the resistance of Internet of Things systems on their data integrity when targeted by False Data Injection Attacks". This question can be divided into several sub-questions, which are expressed below.

RQ1 What are the data gathered and processed by the Internet of Things and what characterises them?
As said above, IoT is about the connection between the physical and virtual world. This connection is made through the data collected by the connected devices. An enormous amount of data are collected by these systems (estimated at 79.4 ZB of data in 2025[4]), so the complete IoT ecosystem is dependent on these data.

To study the FDIA in detail we therefore need to know the nature of the data that passes through the IoT systems. Our objective is in two phases: the first one is the identification and the characterisation of the sensors generally present in IoT devices, and then in the second phase we will be able to identify and characterise the data produced by these sensors. To do this, we will analyse the different IoT systems present within the Flowbird information systems, as well as several other IoT systems or related to them, to achieve the best exhaustiveness.

RQ2 How to model FDIA attacks through Model-Based Testing (MBT) techniques and how generic these models can be within IoT?
With the previous research question, we identify the essence of FDIA, the data ; with this question we use the previous results to propose a Domain Specific Language (DSL) to model FDIA. This comes with two objectives, the first one being to verify the possibility of simulating corrupted data injection type attacks, and the second being to see how generic

[4] https://www.idc.com/getdoc.jsp?containerId=prUS45213219

these models can be. Indeed, although IoT systems have many similarities between them in terms of their architecture, they can differ greatly in regards to their uses and environments, and so the FDIA might be different between IoT.

In order to provide answers, we will study the DSL following a multi-step methodology, which answers two questions. The first is ”Is a DSL suitable for our problem?” and the second is ”How do we make a DSL that fits our problem?”. We will therefore analyse our modelling method through the decision to use a DSL, its application domain, its design, its implementation and its usage.

RQ3 To what extent does the proposed framework address the issue of FDIA in IoT?

The approach proposed in this work aims to improve the handling of FDIA, both in terms of the quality of the systems concerned and the effectiveness of the detection systems, and therefore, to bring a better quality in a preventive way by testing, and in an active way during the operation by detecting. It is therefore necessary to define what our approach brings to this issue. Some sub-questions can then be formulated: To what extent is our approach more effective than currently used methods? To what extent does our approach address the functional issues of FDIA? Is our approach technically efficient enough to be used in real systems in operation?

To answer this question and its sub-questions, we will evaluate the proposed framework through experiments. We will use different IoT systems with different environments and characteristics, and this will allow us to validate our approach through its functional efficiency, its technical efficiency, and its genericity.

1.4/ CONTRIBUTION OF THE THESIS

This thesis proposes a new security testing approach, to address a specific security issue, which can be found in IoT systems: the FDIA. The final objective of this work is to deal efficiently with the FDIA challenge. To this end, we propose a comprehensive step-by-step approach to model FDIA and generate attacked datasets, based on models describing the attacks. These datasets can then be used to address several aspects of the FDIA challenge, such as testing the robustness of systems vulnerable to FDIA, training and evaluating detection tools, or assessing the functional requirements of IoT systems.

The main contributions of this thesis are:

- The definition of a complete step-by-step approach to address FDIA in IoT systems.
- A classification of the sensors generally present in IoT systems, and a characterisation of the data they generate.

- The design of a modelling language dedicated to FDIA, called FD2IOT[5].
- A prototype tool implementing the modelling language, and executing its directives.
- Experimentation of the approach on both objectives, testing and detection, on several case studies.

The following sections present the detailed contributions of the thesis. In addition, we present the research projects in which this work has been involved. The research projects FEDER SARCoS[6] and ANR GeLeaD [7].

1.4.1/ Approach Definition to Address the FDIA Challenge

Our model-based testing approach is a sequence of processes for generating datasets used for test cases or training. Our approach consists of three main processes that can be broken down into sub-tasks:
The first process concerns the IoT data processing. The objective is to retrieve raw data from IoT devices, then configure them according to their characteristics, and finally convert them into a specific format, dedicated to our approach.
The second process concerns the attack design. This is the part where the experts and testers model the attacks they wish to carry out using the dedicated language. It is also in this process that the execution of the modelled attacks is carried out.
The third process concerns the testing. This is the part where the execution results of the previously modelled attacks are recovered. These results are then used either to be injected into the systems under test to perform the planned tests, or used for other purposes, such as improving detection systems.

The different processes of our approach are described in the section 6.1.

1.4.2/ Sensors and Data Classification

To properly model and simulate attacks that specifically target sensors data, we need to know where the data comes from and what characterises them. We therefore focus on identifying the sensors that generally exist within IoT systems, and characterise them according to their functional principle. We give a classification of IoT sensors for FDIA. We then analyse the data produced by these sensors as well as the data that generally transit in IoT systems. This allows us to extract a list of properties that can characterise IoT data, the type or the definition for example.

[5] False Data Injection To Internet Of Things
[6] https://projects.femto-st.fr/sarcos/en
[7] https://projects.femto-st.fr/gelead/en

The sensors and data analysis is carried out in section 4.

1.4.3/ Domain-Specific Language

The approach focuses on modelling attacks in order to simulate them and test their effects on vulnerable systems. To address this modelling need, a domain-specific language has been developed, named FD2IoT for False Data Injection To Internet of Things. It allows the modelling of an injection attack scenario, describing step by step the actions that the attack will have to perform on the data to reach the desired state.
To be precise, the language allows the modelling of an attack scenario based on four alteration primitives used for data manipulation, creation, modification, deletion, and copying.

A complete walkthrough of the language construction through the different phases of its design, as well as its complete grammar, is given in the chapter 6.

1.4.4/ Prototype Tool

A prototype that implements our approach has been developed to conduct our experiments. Developed in Java and using a mongoDB database, the tool implements the DSL and the different processes defined in the approach, and thus importing data, configuring it, designing and executing attack scenarios using the DSL, and finally exporting the attacked data. The prototype, its sources and the language grammar are available at https://gitlab.com/mbriland/fdia-simulation.

1.4.5/ Experimentation

The experiments were conducted on a system present in the company. This system, present inside the parking meters, allows the measurement of the environmental conditions around it, for example, temperature, pollution or noise. We run several FDIA scenarios on these systems to demonstrate both the capabilities of the DSL and the approach. The experiments seek to demonstrate the capabilities of the approach at the level of its expressiveness, to cover IoT systems as a whole, as well as the level of efficiency, to demonstrate the usefulness of the proposed approach in our IoT context. We are also doing some experiments on machine learning based detection tools, to see the effectiveness they can provide, as well as their benefits and drawbacks.
The experiments are presented in the chapter 7.

In the following sections, we present the research projects in which this thesis has been involved.

1.4.6/ RESEARCH PROJECT

This thesis is part of and has contributed to two research projects, both of them focused on the problem of FDIA. The following two sections present these projects.

1.4.6.1/ SARCOS PROJECT

Secure And Reliable Connected Systems (SARCoS) is a European ERDF[8] project which aims to develop intelligent tools to automate the model-based testing of complex connected systems, to find security and reliability problems. In particular, these tools will be used to generate security tests and robustness tests, targeting systems that are vulnerable to false data injection attacks. These tools should:

1. Infer domain-specific test models
2. Generate attack scenarios by learning from execution interaction data
3. Refine and prioritise test cases
4. Produce intelligent analysis of test results

They will be developed in 3 specific areas sensitive to FDIA, thanks to several industrial partners:

- Intelligent Parking Systems (IPS) and IoT with Flowbird
- Air Traffic Control (ATC) management systems with Smartesting
- Secure Timing with Gorgy Timing

This thesis deals with the industrial IPS and IoT domain of this project.
The overall goal of this project is to greatly reduce start-up overheads and testing costs to enable cost-effective testing of interconnected systems vulnerable to FDIA. This will be achieved by providing tools for modelling said systems and attacks, then generating efficient test data from them.

1.4.6.2/ GELEAD PROJECT

Generate & Learn & Detect (GeLeaD) is an ANR[9]-ASTRID[10] project which aims to improve the detection of FDIA attacks by IA components, driven by automatic generation

[8] European Regional Development Fund
[9] National Research Agency
[10] Specific Support for Defence Research and Innovation Work

of tests from attack patterns. Indeed, defence control systems and civilian systems are increasingly exposed to cyberattacks such as FDIA. The use of Machine Learning (ML) for security anomaly detection, malware analysis, and pattern and signature recognition is an extremely active topic in both research and cybersecurity industry. The detection of abnormal signals, as well as the identification of correlations with cyber-attack patterns, are the first applications researched for ML techniques in this context. These techniques are currently developed on low-level traces, and do not address the semantics of business data due to the domain-specific nature of FDIA. For example, a FDIA on a Smart Grid could involve fine modification of data from production nodes, while an attack on a defence air traffic control system could involve falsification of airspace runway data. The other important limitation is the lack of large amounts of representative falsified data, which is crucial for training Machine Learning models, especially when these are based on artificial neural networks.

The GeLeaD project will develop a demonstrator that will be tested in two business areas:

- Civil and military air traffic control on ADS-B protocols
- IoT systems (noise and pollution sensors)

This thesis addresses the latter by providing clean and attacked datasets to address the domain of IoT systems.

1.5/ ORGANISATION OF THE DISSERTATION

This thesis is organised in eight chapters, as following:

- Chapter 1 is the introduction of this thesis, we present the context of the company where the thesis was made, as well as a context of IoT cybersecurity. We also present the research questions and the general approach we have chosen to follow.
- Chapter 2 is dedicated to the presentation of the Flowbird information system. The aim of this presentation is to understand how the company's systems have real security needs and why the analysis and processing of FDIA are essential.
- Chapter 3 is dedicated to the state of the art of Internet of Things security and in particular FDIA.
- The chapter 4 is a contribution about the characterisation of sensors generally present in the IoT domain as well as the characterisation of the data generated and processed there.

- Chapter 5 describes the language developed for modelling FDIA scenarios.
- Chapter 6 describes the framework proposed through the global approach, the tool developed and the use of the DSL.
- In chapter 7 we focus on experiments and use different case studies to demonstrate the validity of our work.
- We conclude this dissertation with chapter 8. It aims to summarise the work done during this thesis as well as to study the future works that can be carried out based on this thesis.

2

Flowbird Information System Security

Contents

2.1 Information System Presentation 14
2.1.1 Parking Meter 14
2.1.2 Network 16
2.1.3 Data Centre 16
2.1.4 Third Parties 17
2.2 Risk Analysis 17
2.2.1 Parking Meter 17
2.2.2 Network 19
2.2.3 Data Centre 19
2.3 Internet of Things Analogy 20
2.3.1 Perception Layer 21
2.3.2 Network Layer 21
2.3.3 Middleware Layer 22
2.3.4 Application Layer 22
2.3.5 Business Layer 22
2.4 Conclusion 22

The company Flowbird has a large information system. Indeed, managing hundreds of thousands of devices around the world with all their associated services requires a lot of resources.

In this chapter, we explore the Flowbird information system related to the parking meter, and thus express the security problems that can occur there. The Fig.2.1 shows the simplified infrastructure of the parking meter system. We can identify two main kinds of data passing through this system: banking data and operational data. These two kinds of data follow separate circuits for obvious security reasons. These data also flows through four main areas: parking meters, the network (all arrows), the data centre (orange dashed

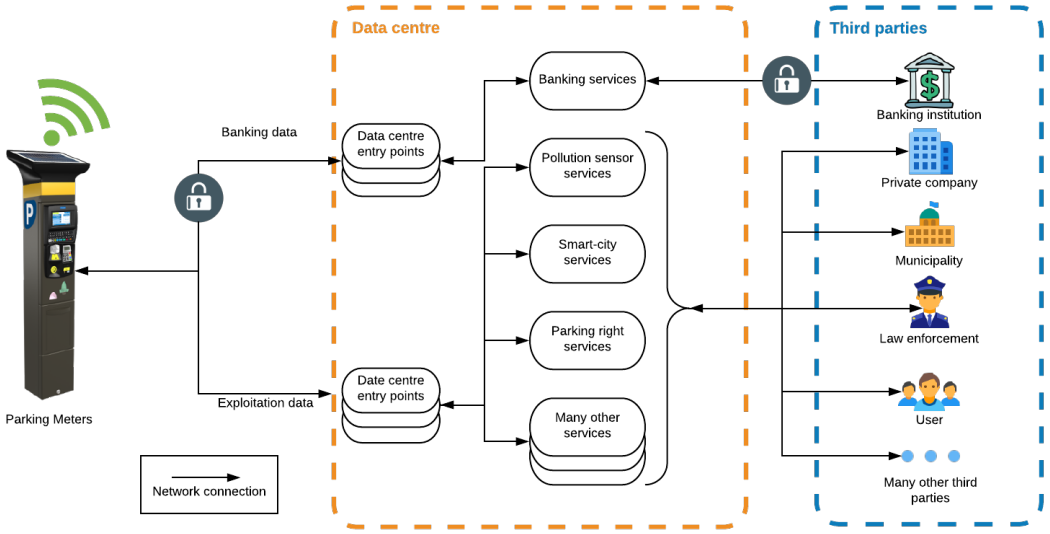


Figure 2.1: Simplified infrastructure of the parking management system

box) and third parties (blue dashed box). We will look in detail at the first three areas, as third parties are not part of the company's systems, they are outside our studies' scope.

2.1/ Information System Presentation

In the following sections, we therefore present the different areas with their data and the risks that may occur. We also present their link with IoT and the high dependency on the collected data. We then conclude by identifying the challenges to be addressed.

2.1.1/ Parking Meter

The lowest level in the information system is the parking meter. Initially devoid of any intelligent systems, connections or services other than pure parking time measurement, these devices have evolved gradually to become what we know today in our streets and continue to evolve even today.

This evolution has been mainly incremental, adding different systems and services in response to user needs and technological advances. The very first parking meter was a machine with the unique ability to take money to start a parking time countdown. The only peripherals of these machines were the coin acceptor and the timer. At that time there was one machine for one or two parking spots. Quickly, the number of machines was reduced to handle more parking spots per parking meter. With this change in functionality, changes within the machine also took place, particularly in the parking right issuing system. For a long time, the parking ticket remained a simple paper ticket to be displayed

in the vehicle. Still in use today, this paper is disappearing to be replaced by more modern methods, notably the use of the licence plate number for identification. The second important change was the introduction of new means of payment. This is the emergence of traditional bank card payment, followed by contactless payment (bank card, smartphones, smart watches, etc.). The emergence of different pricing levels has also led to the development of new services such as subscription systems. All these changes have obviously led to the connection of the parking meters to the internet, notably through mobile communication protocols such as GPRS or LTE, as well as the development of numerous peripherals. The Operating System (OS) has also greatly evolved. At the beginning it was a very simple and light proprietary OS, running on a very low power microcontroller. It has gradually evolved to become more complex as technology and needs have changed. Today, the new generation of parking meters come out with Android as OS. All this evolution in technologies, such as network connections and peripherals, has allowed the appearance of many specific services, no longer reducing the parking meter to a simple timer, but to a complete connected object offering many functionalities. Nowadays, parking meters are sometimes called multi-service kiosk. All these changes have allowed the evolution from a simple system with a few peripherals and functionalities, to complex systems with dozens of them, depending on the configuration. The parking meter is therefore the interface between the users and the parking operator. It is in charge of data collection and restitution. A typical parking meter in 2021 could be described as shown in Fig. 2.2.

Figure 2.2: 2021 typical parking meter

2.1.2/ NETWORK

As presented in the previous subsection, the parking meters have gradually become connected. Most of them are now connected to the internet in order to offer more complete services. This connection, although existing through traditional cable networks, is mainly achieved through mobile networks, in particular GPRS protocol for the oldest and UMTS, HSPA, HSPA+ or LTE for the most recent.

The network is used for two main purposes: an ordinary connection for operating data and a banking connection for banking transactions and data. All connections coming out of the parking meter go through the data centre. This two different transaction types don't rely on the same security measures and don't follow the same path through the network. All the banking transaction must follow the Payment Card Industry Data Security Standard (PCI DSS). These transactions are encrypted within the card readers and are processed in a completely different way.

The vast majority of parking meters are battery-powered, which are recharged by a solar panel. Networking must therefore be as energy-efficient as possible. In the early days of the development of connected parking meters, the main protocol used for network communication was developed directly within the company to meet the need for energy and network savings. Nowadays, the parking meters use proven and reliable protocols such as MQTT for their communication.

The network is therefore responsible for transporting all types of data in both directions, between the parking meters and the data centre.

2.1.3/ DATA CENTRE

All data and transactions from parking meters around the world are sent to the company's data centre. Depending on which parking meter generation and which different services are used, a lot of data passes through the data centre and is used in many different processes. These processes and services can be, for example, parking meter management, subscription management, parking fines, pricing grids, environmental conditions, etc.

As with the other areas, developments have evolved over the years, starting with services using ad hoc technologies and protocols. The services have gradually evolved towards the use of widespread and open source technology such as elasticsearch to handle the huge amount of data retrieved every day. The data centre also exposes these services to a large number of third parties.

The data centre is therefore the heart of all parking management activity. All data, retrieved by parking meters or sent by third parties, passes through it.

2.1.4/ THIRD PARTIES

Many third parties have access to the data centre services. Third parties are outside the scope managed by the company, we will therefore not develop this part. Nevertheless, third party access to the applications offered in the data centre is very extensive. We must keep in mind that security within the data centre must be strong in both directions: a strong authentication of third parties for their connection to the data centre services, and a strong integrity of the services provided to the third parties for business purposes. Third parties can access many services such as parking control services, which can use simple smartphones as well as Automatic Number-Plate Recognition (ANPR) systems linked to the data centre databases.

2.2/ RISK ANALYSIS

In the previous section, we described the general information system related to parking meters. Now we will see how these systems can be vulnerable by performing a risk analysis on this environment. Risk analysis is necessary to know the threats and vulnerabilities of a system. In an industrial context such as ours, it is necessary to identify vulnerabilities in order to address them if necessary, so all work is carried out on a risk analysis basis. This allows us to motivate the importance of addressing the FDIA challenge.

This analysis was done on several levels, including in-depth analysis of the systems, discussion with the various project managers and modelling of the systems. For example, by using the securiCAD modelling tool [Ekstedt et al., 2015]. By modelling the IT environment, modelling the attacker and using attack graphs, this tool can calculate, analyse and highlight potential vulnerabilities of the modelled system. The modelling in this tool of the IT environment of one service present on parking meter, is shown in Fig. 2.3. Here we can see the modelling of an attacker specifically targeting the network between the different entities of the system. Also we can distinguish the three parts (parking meter, network, and data centre) belonging to the enterprise. The risk analysis of these parts is detailed in the three following sections.

2.2.1/ PARKING METER

Parking meters are devices that can be found on every street around the world. The first threats to them come from their direct environment. These are physical threats, including sabotage, destruction and disruption of service, malicious or non-malicious. Indeed, this type of threat can be caused by an ill-intentioned individual with the sole purpose of harming the devices, or by natural disasters or accidents. These threats are countered

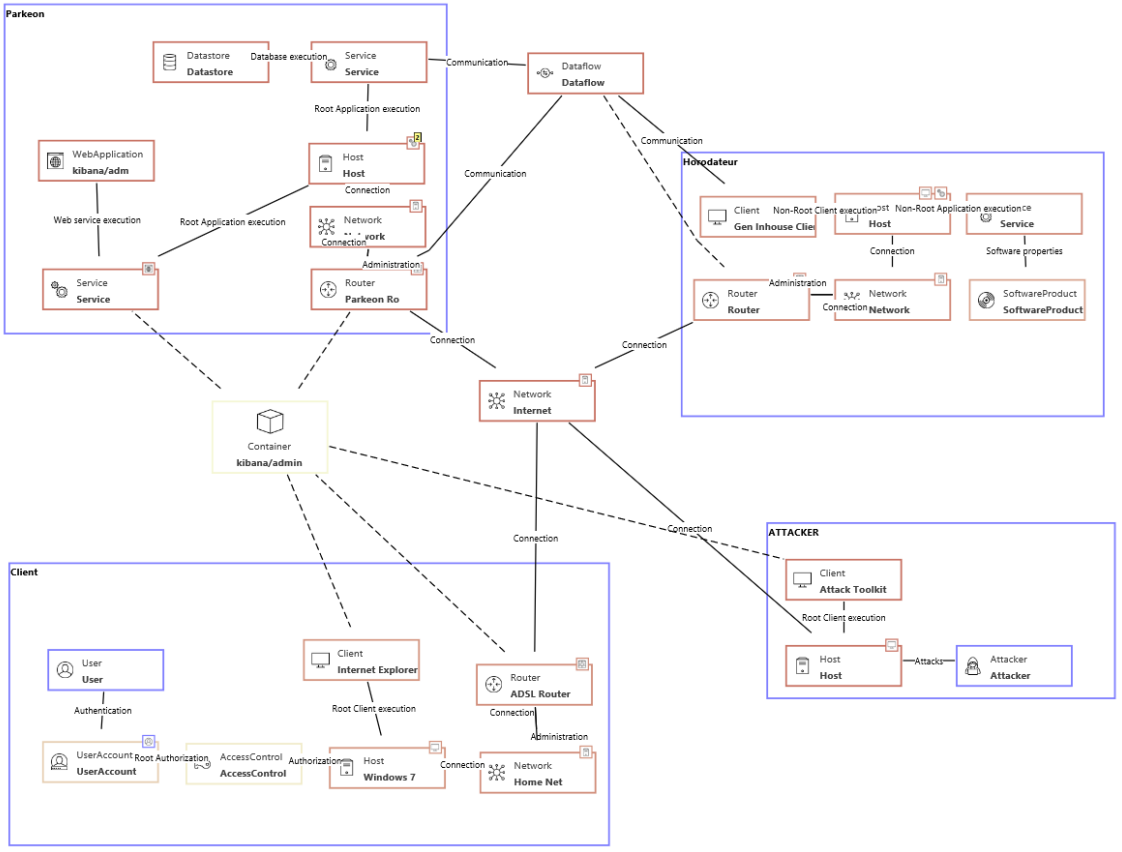


Figure 2.3: securiCAD modelling of one IPS services for risk analysis purposes

by the robust construction of the parking meters. In particular, they are subjected to numerous physical attack tests and are submitted to strongbox certification.

Another critical risk to parking meters is the bank data circulating in them. These risks are regulated by stringent standards that must be met by entities that have to process banking transactions. These include the PCI DSS or the Payment Application Data Security Standard (PA DSS). An external certification body is responsible for ensuring annually compliance with standards. All bank connections are encrypted directly in the card reader before being sent over the network.

Taking over the parking meter constitutes the majority of the remaining threats. The major risk involved is the consultation or modification of the data that is silently transmitted through it. A malicious person having access to the internal systems of the parking meter can act on all the data transiting through it. Whether it is to shut down the device, change the pricing rates or send erroneous data to the data centre. However, this problem is mitigated by the obscurity of older parking meter technologies (see section 1.1). Newer systems are now implemented with newer technology that is generally open source, with a huge community of contributors and users, such as the Android operating system. They should therefore benefit from advanced penetration testing, intrusion detection systems

and constant vulnerability monitoring (especially for updates) to reduce the risk.

2.2.2/ NETWORK

On the parking management system, we can distinguish three different networks in our architecture. The mobile network between the parking meters and the data centre, the data centre network and finally the network for connecting to third parties. These three networks are like any network around the world. They can be subject to the same type of attacks, such as Denial of Service (DoS) or eavesdropping attacks. The network, in the case of parking meter use, is highly sensitive to one type of threat. These are threats to the integrity of the data it carries. Indeed, attacks affecting service availability such as DoS attacks have a limited impact: parking meters can operate in degraded modes without a network for a certain period of time. Eavesdropping attacks also have a limited impact, as the data in transit contains little critical information and the critical information are encrypted. In contrast, attacks that alter the data are truly critical to the system. Attacks such as Man-In-The-Middle (MITM) allow an attacker not to only listen to the communication on the network but also to modify them. Such modifications can lead to serious services disruption. The data centre, which is the heart of all the services provided by parking meter, must be able to rely on reliable data for providing a good quality of services.

2.2.3/ DATA CENTRE

The data centre is central to the overall system operation. It hosts a multitude of different services and gives access to them to a multitude of different third parties. This large number of entities gives it a very large exposure surface.

Among these entities, many proprietary services coexist alongside many open, widespread and off-the-shelf services. These widespread services bring with them many vulnerabilities. Since they are widespread and the source code is often available, many people, malicious or not, can discover vulnerabilities. This makes services vulnerable for a short period of time after the discovery of the threat. Thanks to their large community, fixes to these vulnerabilities are often very fast to appear. This brings a very high level of security, if a constant technological watch on the relevant tools is set up, with a fast application of security patches. For proprietary services, the situation is different. Because there is no active public users community forms around them. This implies that many vulnerabilities may exist but may not be detected or fixed yet. This can be mitigated in several ways:

1. Having a policy of security by design from the beginning of the development.

2. Set up unit, functional and penetration tests throughout development and during the life of the product.

3. Have a strong security policy for the configuration and management of the infrastructure hosting the services.

In any case, for the optimal functioning of all the company's activities, the data centre must be completely secure, especially in the area of data management. Data integrity in the data centre is essential to the functioning of the entire chain, as it is in the two other parts of the architecture.

2.3/ INTERNET OF THINGS ANALOGY

A fairly strong analogy can be made between parking management systems and IoT.

Firstly, on the definition itself. The IoT is a network of networks, it is composed of many heterogeneous devices things and technologies. The main idea behind the IoT is to provide an interface between the virtual world and the physical world, allowing the recovery, transfer, storage and processing of the data gathered by the things to provide services. The parking management system follows the same definition. We have heterogeneous devices on the street who gather real world information such as parking time or pollution level and transfer them to the virtual word to provide specific services.

Secondly, on the intrinsic characteristics and properties of parking meters. They are autonomous, have on-board computing power and a network connection. These are three characteristics that are generally found in IoT devices.

Thirdly, on the architectural level. IoT is basically described as a five-layer architecture [Miao Wu et al., 2010] in the literature. The perception layer, the network layer, the middleware layer, the application layer and the business layer. This architecture of the Internet of Things will be presented in more detail in the state of the art, see chapter 3. Although services related to the middleware, application and business layers are mainly located within the data centre for parking systems, a strong analogy can be made about the architecture. Fig. 2.4 compares the theoretical architecture of an IoT system with those of a standard IoT system and a standard parking system.

Just like the Flowbird systems, IoT systems need an extremely high level of confidence in the data that is retrieved by the objects. If data integrity is not ensured on one layer, the poor integrity will spread over to all the upper layer (e.g., a failure in integrity on the perception layer will lead to a failure in integrity on all the other layers of IoT architecture). In the following, we see the different IoT layers and how they compare with parking meters. We also see why the data are so important.

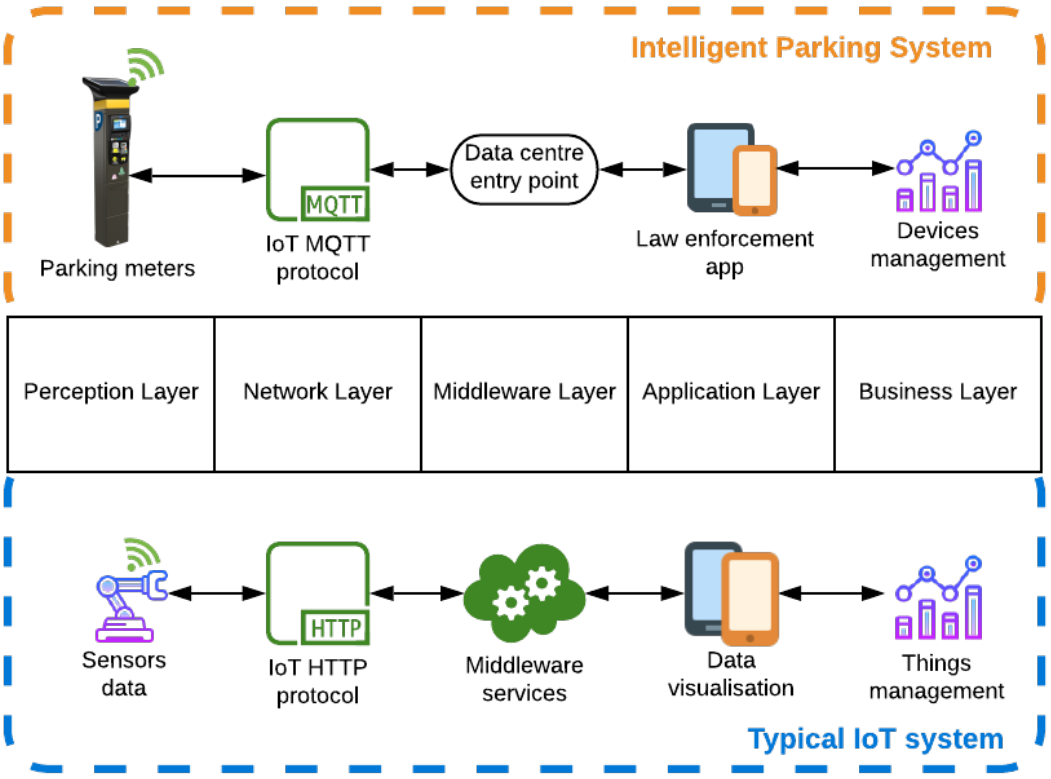


Figure 2.4: Analogy between the architecture of an IoT system and an IPS

2.3.1/ PERCEPTION LAYER

On IoT the perception layer is where the devices are located (sensors and actuators). It is the ”things” of IoT. This layer is the physical part of the IoT, it is essentially about the data collection by the sensors. That’s why we can make an analogy between a parking meter and an IoT device and put it under the perception layer in Fig. 2.4. Like any IoT device the parking meter gather real world data, and then transmits it to a network. Next, we move on to the network layer.

2.3.2/ NETWORK LAYER

The purpose of the network layer is to transport the data collected by the elements present in the perception layer. It allows communication between all layers of the architecture. In IoT systems the network layer uses many different protocols and technologies depending on their situation. For example, an IoT device in a house used to monitor temperature will usually communicate via WiFi and use HTTP requests. For parking meters the network is used in the same way, it usually uses mobile networks such as LTE to communicate

and MQTT as protocol. The data are now transmitted to the middleware layer.

2.3.3/ Middleware Layer

As said before, IoT systems are extremely heterogeneous. Numerous sensors work side by side with numerous actuators, using many different protocols and technologies. The middleware layer aims at aggregating all this heterogeneity. It is also responsible for the data processing and service management. In the case of parking meters, aggregation services are used as an entry point into the data centre. They make it possible to cope with the heterogeneity of versions and communication systems of parking meters. They also processes and redirect the different connections to the right services. This is why we can classify them within the IoT middleware layer.

2.3.4/ Application Layer

The application layer is the front end of the data processing, it is where the information of all the data gathered by the perception layer is displayed to the end user. In the case of parking meters, the application layer includes many different services. For example, there is an application to display whether a car parked on the street has the correct parking rights, as part of parking control.

2.3.5/ Business Layer

The business layer aims to manage all the IoT system, it is usually used to analyse the system and improve it, whether it is a technical improvement or an improvement of the business model. It is also responsible for managing user privacy. In the case of parking meters, these are specific services that directly manage the underlying business. It is located in the same places as the services of the application layer. They just don't have the same functionality.

2.4/ Conclusion

In this chapter, we analysed the company's information system around parking management systems. We have seen that it is divided into three main areas, the parking meters, the network and the data centre. These areas, depending on their nature, have many associated risks. We have also seen that the parking management system can be compared to an IoT system. Indeed, their architectures, their security problems and their high dependency on the data circulating in them, highlight a great similarity between the two

domains. Regarding the data, we have identified two main types of data that flow through these areas: banking data, which is encrypted, and operational data. All this circulating data are very important for the company. Trust in data and in their integrity is essential, as data are at the heart of the services offered by the company and therefore of its business. Following this, in the continuation of this dissertation, we will consider the field of the IoT as the main field of research, based on the systems proposed by the Flowbird company. In the next chapter, we look at the state of the art in IoT security and in particular the handling of FDIA.

3

State of the Art

Contents

3.1 Internet of Things Security 26
3.1.1 Perception Layer 27
3.1.2 Network Layer 28
3.1.3 Middleware Layer 28
3.1.4 Application Layer 29
3.1.5 Business Layer 30
3.2 False Data Injection Attack 30
3.2.1 FDIA Definition 32
3.2.2 FDIA Research 34
3.2.3 Related Works 35
3.3 FDIA Testing 37
3.3.1 Model-Based Testing 37
3.3.2 Data Generation 39
3.4 Conclusion 43

In this section, we review the existing works on IoT security and testing. As mentioned earlier, IoT systems are relatively new. They benefit from constant technological evolution and increasing adoption in our society. As a result, research in this area is really prolific, accompanying this evolution and adoption. A lot of research is being carried out, especially in terms of security issues. First, we will look at IoT through its architecture and the security problems existing within its different layers. This will lead us to underline the omnipresence of FDIA within the IoT layers, as well as the importance of the security issues they can bring. In a second step, we will study the FDIA, first by giving it a definition based on the descriptions given in the relevant literature. We will then analyse the current state of research on FDIA, particularly in the various fields targeted by FDIA. We will next find works that may be related to ours, and see how our problems are addressed in them. Finally, in a third step, we will study the options available to us to be able to test FDIA, in particular by analysing existing test methods, as well as data generation methods.

Architecture layers	Security issues
Perception layer	Hijacked/fake node, False Data Injection Attack, Denial of Service (DoS), Authentication attack, Cloning, Eavesdropping, Spoofing, Jamming, Inference attack, Physical attack
Network layer	State-of-the-art security attack. e.g. Sybil attack, Sinkhole attack, Sleep deprivation attack, DoS attack, False Data Injection attack, Man in the middle attack, Traffic analysis, Routing attack, Selective forwarding
Middleware layer	Unauthorized access, DoS attack, Malicious insider, False Data Injection
Application layer	False Data Injection, DoS attack, Spear-Phishing attack, Sniffing Attack
Business layer	Same as the application layer

Table 3.1: Architecture layers and their security issues

3.1/ INTERNET OF THINGS SECURITY

Since the beginning of research on IoT security, many solutions have improved the overall security of the IoT chain. However, with the constant evolution of the technologies used, and their massive adoption all over the world, many problems still need to be solved. We can divide the IoT systems according to their architecture, which allows us to identify the existing security issues on each of the layers that compose them. IoT architecture has been a much studied subject and has evolved a lot since the early 2000s. Despite the efforts of many organisations, there is no standard architecture today. Many architectures have existed with the different needs that IoT has brought, such as the three-layer architecture, the middleware architecture, the service-oriented architecture, the five-layer architecture, or the cloud-specific architecture [Aswale et al., 2019]. Nowadays, the most widely used architectures in literature are layered ones and they generally consist of 3 to 5 layers, depending on the requirements [Belkeziz and Jarir, 2016]. They were at first described with three layers: the perception layer, the network layer and the application layer. The architecture can be extended with the addition of a fourth layer [Sikder et al., 2018] and a fifth layer [Khan et al., 2012] [Miao Wu et al., 2010] [Aazam et al., 2014], respectively the middleware layer and the business layer. As the architecture with five layers is the most detailed and can always be refined into a lesser layered one, we will base ourselves on the one presented in the previous chapter (see Fig. 2.4).

In these following sections, we provide for each layer some examples of security aspects [Farooq et al., 2015; Zhao and Ge, 2013; Padmavathi and Shanmugapriya, 2009], and then we explain why the IoT is vulnerable to False Data Injection Attack (FDIA). Table 3.1 summarises the explored attacks, classified in their respective layers.

3.1.1/ PERCEPTION LAYER

The perception layer is the physical layer of IoT. First, it can be threatened due to its vulnerable physical condition, being composed of many heterogeneous data sensors, often easily accessible. A physical threat (malicious attacker or natural disaster) can easily disrupt the functioning of the perception layer. In addition to the physical fragility of its things, the perception layer has to deal with cyber security issues. Whether it is hijacked nodes, FDIA or DoS, all these threats rely on the poor constitution of things. Due to their short autonomy and low computing power, they often lack proper authentication/encryption mechanisms.

Thanks to all these sensors, it is in this layer that the data passing through the IoT systems are generated. It will be necessary to study the sensors and the data they produce to characterise the information and obtain metadata on the IoT systems.

[Sikder et al., 2018] presents a survey on sensor-based threats in IoT. They have identified four principal threats: information leakage, transmitting malicious sensor patterns or commands, false sensor data injection, Denial of Service.

Information leakage threat occurs when one or more IoT system sensors reveal sensitive data collected by the sensors or on the IoT itself. This type of threat is usually carried out through eavesdropping [Schlegel et al., 2011] or inference attacks [Cai and Chen, 2012].

Transmitting malicious sensors commands threat occurs when a sensor is used for transmitting and triggering special commands or action. It can create special, unintended communication channels if malware is present on the device [Uluagac et al., 2014]. This kind of attack is called side channel attack.

False sensor data injection is an attack where a malicious adversary alter the data gathered by the sensors. Attackers can have multiple objectives, which mainly aims to modify behaviours within the sensor itself but also much further in the IoT chain. As mentioned in the previous chapter, IoT depends largely on the collected data and therefore on the trust that can be placed in them, this attack is therefore a real threat to the operation of IoT systems.

Denial of Service is self-explanatory, it is a type of attack where the attacker tries to disrupt the service provided by a device or an application. In the case of sensors, such attack can have multiple impacts, for example abnormal battery consumption, but the major one is the direct disruption of sensor services, i.e., the sending and receiving of data.

3.1.2/ NETWORK LAYER

The network layer is the transportation layer of IoT. Its purpose is to transmit data, notably between the perception layer and the middleware layer, but also between the other ones. The IoT network layer is the same as in all network layers in other IT domains. The big difference lies in the great heterogeneity of technologies used by the IoT platforms: there are many different ones, such as WiFi, LTE, Zigbee, or LoRaWAN. The IoT network layer is the same as that of other networks, hence it also faces the same security issues as ordinary networks. We can therefore find a very large number of possible attacks, such as sybil attack, sinkhole attack, sleep deprivation attack, DoS or FDIA.

[Zhao and Ge, 2013] presents a survey on the IoT security. They have identified four main categories of threats: traditional security problems, compatibility problems, cluster security problems and privacy disclosure problems.

Traditional security problems are the problems found in any other network around the world. They will mainly threaten data integrity and confidentiality. Thanks to extensive research over the years, communication networks have rather comprehensive protection measures. However, threats still exist, such as eavesdropping attacks, Man-In-The-Middle attacks or DoS attacks.

Compatibility problems come from the fact that the IoT is especially oriented for machine-to-machine communication, and not only for machine-to-person communication. The great heterogeneity and the need for interoperability also make this compatibility complicated and many security vulnerabilities can then appear.

Cluster security problems are the issues posed by the very large number of devices that can constitute an IoT system, especially in terms of authentication issues. The very large number of devices makes the usual authentication systems unsuitable, and could lead to network service disruption, especially in case of mutual authentication between numerous devices in the same network.

Privacy disclosure is defined as the category where all new means of information retrieval technology and social engineering are involved. Attackers can easily gather a large amount of user privacy information in the network layer.

3.1.3/ MIDDLEWARE LAYER

[Ngu et al., 2016] defines the middleware layer as the intermediary between the IoT devices and the application. It aims at a seamless integration between the physical world and the virtual one. [Khan et al., 2012] defines this layer as the service manager. It owns the links to the database and receives information from the network layer and stores it in databases. It can also perform computations and make decisions based on this informa-

tion.

[Fremantle and Scott, 2017] identified 22 IoT middleware systems and reviewed them on their security model and/or security implementation. They identified four flaws in IoT middleware systems: (1) No explicit concept of privacy by design, (2) No concept of context-based reputation or security for IoT devices, (3) No user-centric access control model, (4) No federated identity system at the perception layer.

[Ngu et al., 2016], on their observation, define three types of IoT middleware architecture. *Service-based* is usually based on a Service-Oriented-Architecture (SOA) and then consider IoT devices as services. *Cloud-based* limits the number of IoT devices deployed but let users collect and interpret data easily; possible use cases can be identified and implemented on the collected data. *Actor-based* prioritises open and interoperable architectures: IoT devices are seen as reusable and distributed actors within the network. In fact, all these architectures use the cloud to host their services, so the majority of security issues within this layer are the same as those of traditional cloud services. [Fremantle and Scott, 2017] states that the difference with traditional security issue is on the confidentiality, the authentication and the access control levels. IoT adds to the traditional middleware security challenges the privacy concerns, the lack of standards around device identity, and the requirement for users access control.

[Farooq et al., 2015] define three threats for the middleware layer. The *Unauthorised access* threat could be fatal for the system: this layer offers multiple interfaces, and if an attacker access the system, he can then perform multiple attacks, such as False Data Injection Attack, or the disabling of IoT devices. *DoS attack*, as in the two previous layers, aims at disrupting the services. *Malicious insider* is an attack made by a person with authorised access to the system, potentially causing fatal damage to the system, as in the case of unauthorised access. In general, this type of attack is intended for personal gain or ordered by a third party.

3.1.4/ Application Layer

This layer is the interface of the IoT architecture. It provides all the services requested by the users, it is where the data interpretation takes place. The application data of this layer comes from information stored and processed in the middleware layer. The application layer covers a huge field of applications, such as smart health, smart home, smart transportation [Choudhary and Jain, 2016] or, of course, smart parking. On the security level, this layer must address the same security issues as all application types (web application, mobile application, desktop application, etc.). Usually they are attacked to gain access to the application, to gather or modify data, such as in sniffing attacks or malicious code injection attacks.

In [Farooq et al., 2015], they identify four security challenges for the application layer: malicious code injection, DoS, spear-phishing attack and sniffing attack. This identification is mentioned in [Swamy et al., 2017] and the authors add four security-related problems: authentication of identity, data storage and recovery, handling huge data, application layer software vulnerabilities. In their survey, [Hassija et al., 2019] identify six major security issues which encompass the precedent ones: data thefts, access control attacks, service interruption attacks, malicious code injection attacks, sniffing attacks and reprogramme attacks.

3.1.5/ Business Layer

This layer is the last to appear. It is a specialisation of the application layer. This specialisation is about the management of the entire IoT system, it manages both applications and services, and allows the business model to be defined through graphs and flowcharts, based on the data retrieved from the applications [Farooq et al., 2015]. This layer is also responsible for the users' privacy, and has the responsibility of the data management (creation, storage and modification) [Burhan et al., 2018].

The business layer being a specialised derivative of the application layer, it will therefore be subject to the same types of attacks. The difference lies in the attacker's willingness to impact on the business or the functional aspects of the IoT system. This layer will therefore be susceptible to attacks such as malicious code injection, FDIA, or DoS.

3.2/ False Data Injection Attack

As we saw in the previous sections, IoT systems have many threats on them requiring work to secure them. Nevertheless, a particular type of attack is notably present within all the architectural layers of the IoT: these are attacks designed to break the data integrity circulating within IoT systems. One of these attacks is the FDIA; in this section we will see what FDIA is, and what is the state of research on FDIA globally, and then more specifically in the IoT domain.

FDIA were described at first in the domain of sensor networks [Sencun Zhu et al., 2004] and later more widely on the Wireless Sensor Network (WSN) [Ma, 2008]. In particular, the research was very active in the energy field, for the smart grids [Liu et al., 2009; Xie et al., 2010; Liang et al., 2017b].

A WSN is a network of nodes, often sensors, with very low and limited computing power, that send the data they retrieve through wireless protocols such as the IEEE 802.11 families of standards [Dargie and Poellabauer, 2010]. The data are sent to a base station,

also known as a sink. This communication can occur directly from a node to a sink but can also occur through multiple nodes to reach a node close enough to a sink. Based on the data received, the base stations perform calculations and can make decisions. A typical FDIA scenario on a WSN is an attacker who takes control of one or more nodes on the network and submits erroneous data, instead of the data collected by the nodes. These false data are generally not random, as if they were just interfering with the service, but have a deeper impact on the base station's decision-making.

In the smart grid domain, which is the most abundant in FDIA research, the FDIA is a type of attack where the goal of the attacker is to introduce arbitrary errors on the state variables of different meters, these errors leading to a corrupted state estimation. The state estimation is when a smart grid uses the different reports of all the power sensors to perform an estimate of the power consumption and adjust the power production in real time. If the state estimation receives corrupted data, it can lead to erroneous assumption of current power consumption and cause a complete interruption of power distribution, and power outages. An example of countrywide FDIA took place in 2015 in Ukraine [Liang et al., 2017a] where an attack compromised six organisations, including three electric distribution companies. The attack resulted in power cuts throwing part of Ukraine in a blackout and affecting approximately 225,000 Ukrainians [ICS-CERT, 2016].

Broadly speaking, FDIA is an intentional (malicious) or unintentional attack that compromises a sensor network, usually by taking advantage of one or multiple sensors. The aim is to generate and send data or events that don't represent the reality of the network environments. This type of attack can lead to many types of issues depending on the targeted domain, such as energy and network waste, money loss, and even physical destruction [Albright et al., 2010].

In the case of unintentional FDIA, there is no attempt to harm a specific system, and therefore no attacker. It is more likely a False Data Injection (FDI). Usually, it happens in the perception layer and is an anomaly in the sensor environment. It can also be an error inside the sensor itself, such as sensor failure or malfunction, notably because of the resource-constrained nature of sensors and the unreliability of their networks [Hamdan et al., 2012]. Even in this case, these FDI still need to be protected from, to prevent damage and disruption of the system. An example of unintentional FDI can be a worm on a humidity sensor that then stops watering a crop, a lorry parked next to a pollution sensor that triggers a specific event, or a malfunctioning thermal sensor that triggers a fire alarm.

3.2.1/ FDIA DEFINITION

As far as we know, no one has established a real definition of what an FDIA is. Most researchers describe the FDIA in terms of what it should do in their specific context and do not always use the term FDIA to define this type of attack behaviour; it can also be found under the term of data integrity attacks, stealthy integrity attacks, deception attacks, etc. In this subsection, we review some ”definitions” from several articles in several domains such as smart grid or Cyber Physical Systems (CPS), as shown in the table 3.2, and then attempt to provide a comprehensive generalist definition.

Author	**Domain**	**Description**
[Sencun Zhu et al., 2004]	Sensor network	”an adversary may compromise several sensor nodes, and then use the compromised nodes to inject false data[...]. Standard authentication mechanisms are not sufficient[...], since the adversary knows all the keying material possessed by the compromised nodes. We note that this attack can be launched against many sensor network applications[...].”
[Ma, 2008]	WSN	”The type of attack when the compromised sensors forge the events that do not occur.”
[Liu et al., 2009]	Smart-grid	• ”By taking advantage of the configuration information[...], an attacker can inject malicious measurements that will mislead the state estimation process without being detected by any of the existing techniques for bad measurement detection.” • Two goals: ”Random fdia” and ”targeted fdia” • ”fdia do pose strong requirements for the attackers.[...] know the configuration of the target”
[Mo and Sinopoli, 2010]	CPS	”deception attacks (or FDIA) [...] affects the data integrity of packets by modifying their payloads.[...] We feel that fdia can be subtler than DoS attacks as they are in principle more difficult to detect and have not been thoroughly investigated.”

Continued on next page

Table 3.2 – *Continued from previous page*

Author	Domain	Description
[Qingyu Yang et al., 2014]	Smart-grid	”With compromised devices in the grid, the adversary can inject false measurement reports to the controller. This causes the controller to estimate wrong system states, posing dangerous threats to the operation of the power grid system.”
[Cretin et al., 2018]	Air traffic control	”attacks require a deep understanding of the system, its protocol(s) and its logic, to covertly alter (by injecting falsified beacons and deleting genuine ones) the consensus reached by ground stations and aircraft[...]. Such attacks are much more complex to achieve than e.g., jamming because the logic of the communication flow must be preserved.”
[Bostami et al., 2019]	IoT	”FDIA is a cyber attack where the compromised host construct events which do not take place in that instance of time.[...] the attacker can take advantage of the small error rate tolerated by the system algorithms[...]. FDIA requires strong analysis of the target system [...] the attacker must know the topology of the system. Another requirement for FDIA is that the attacker should have physical access to tamper with the system. FDIA has major impacts on the system.”
[Mode et al., 2019]	IoT & Machine Learning (ML) & Predictive Maintenance(PdM)	”In FDIA, an attacker stealthily compromises measurements from IoT sensors (by a very small margin), such that the manipulated sensor measurements bypass the sensor’s basic ’faulty data’ detection mechanism and propagates to the sensor output undetected. An FDI attack can be implemented by compromising physical sensors, sensor communication networks, and data-processing programs. Such attacks on a PdM system may not even show their impact. Instead, the attack propagates from the sensor to the ML part of the PdM system and fools the system by predicting a delayed asset failure or maintenance interval. This might incur a significant cost by inducing an unplanned failure or loss of human lives in safety-critical applications.”

Continued on next page

Table 3.2 – *Continued from previous page*

Author	Domain	Description
[Ye and Zhang, 2020]	CPS	”FDIA intend to break the system performance or stability without being detected only by injecting the additional data. Hence, its design and detection problems are particularly difficult due to the attack objective”

Table 3.2: Summary of FDIA definition

From all these descriptions given by authors from multiple fields, we can highlight several elements characterising FDIA. The first element is the very nature of the attack: its purpose is to inject forged and corrupted data into a system. The second element is the prerequisite for the attack, everyone agrees that the attacker must have a very deep understanding and knowledge of the system. Another precondition is the compromising of the system, in particular the data retrieving parts such as nodes in WSN or things in IoT. The third element is the will of the attacker, their attack does not aim directly at stopping the services provided by the system under attack, but rather at modifying them through injection in a subtle way. Hence the need for in-depth knowledge of the systems. This attack also has a stealth purpose, as the attacker seeks to pass through the various intrusion detection systems and filters, notably by using the tolerated margin of error of these systems.

Definition 3: False Data Injection Attack

The FDIA is a cyber-attack where an attacker, thanks to his in-depth knowledge of the system under attack, compromises it, notably at the level of data production devices and data manipulation software. The objective is to inject falsified and altered data, with the aim of modifying the normal behaviour of these systems. It is a discreet attack that performs small injections, in order to be undetectable by the various protection systems: the injection can therefore extend over a long period of time, and is difficult to detect.

3.2.2/ FDIA RESEARCH

Since the first appearance of the term, research on the FDIA has mainly focused on the impact, filtering and detection of these attacks in multiple domains, with a strong focus on smart grid. In Google Scholar, when you search for tittle publication with the first query in Fig. 3.1, 490 results are returned.

When we decide to focus our search in the IoT domain with the query in Fig. 3.2 only 5 results are returned.

```
allintitle:
"false data injection attack" OR "false data injection attacks"
OR "stealthy injection attack" OR "stealthy injection attacks"
OR "bad data injection attack" OR "bad data injection attacks"
OR "injected false data"
```

Figure 3.1: Query 1: Publication where the title contains terms covering FDIA

```
allintitle:
"false data injection attack" OR "false data injection attacks"
OR "stealthy injection attack" OR "stealthy injection attacks"
OR "bad data injection attack" OR "bad data injection attacks"
OR "injected false data"
"Internet of things" OR "Internet of thing" OR iot
```

Figure 3.2: Query 2: Query 1 + containing the terms covering IoT

By modifying this last query to look for the popularity of different domains in FDIA research, we can see in the table 3.3 that IoT is a very little-explored domain as of today. The terms used for queries are, of course, not exhaustive. Publications may escape these queries because they use synonyms, or do not explicitly refer to the domain in the title. Nevertheless, we have browsed through the 490 articles presented in query 1 (Fig.3.1). We can therefore say that the table 3.3 has a good representation of the publications relating to the domain. Other terms can for example cover the field of smart grids, such as SCADA or state estimation. However, these keywords can also be found in other domains such as WSN, so we do not use them in our queries. Other areas also appear in the results of the first query and are poorly explored, so they do not appear in our table, such as multi-agent systems or health care.

[Bostami et al., 2019], in their review of FDIA in IoT domain, came to the same conclusion. The FDIA is a big challenge within the IoT, but apart from smart grids, it has not been studied a lot.

As we have seen, the FDIA is a field that is quite popular and well studied, nevertheless its studies focus on a few very specific areas, and so they require further research in less studied areas. Moreover, by deepening the exact research topics, we can see that a majority of the work focuses on topics of detection and filtering of these attacks. We must now study in this mass of work those that can be related to our concerns, and those that can also be associated with the IoT ecosystem.

3.2.3/ Related Works

As mentioned above, the FDIA is a subject of much study in certain areas, particularly in the detection and filtering of these attacks. One of the difficulties of this work domain ,is

Domain	Query terms	result number
Smart-grid	”smart-grids” OR power OR ”smart-grid” OR electricity OR AC OR DC	254
CPS / CPNS	CPS OR CPNS OR ”cyber physical” OR ”control systems” OR ”control system”	81
WSN	WSN OR ”wireless sensor network” OR ”wireless sensor networks”	49
IoT	”Internet of things” OR ”Internet of thing” OR iot	5
ATC	ADS-B OR ATC OR ”air traffic”	4

Table 3.3: Query on Google Scholar of different domains associated with FDIA

to be able to develop, train and verify filtering and detection techniques, using real data from systems in operation, that have already been attacked. In general, the attacked data are either protected for confidential purposes or simply not detected and therefore not reported as compromised.

To develop their attack mitigation systems and validate them through experimentation, the various authors used several methods to generate data. [Yang et al., 2017] uses pseudorandom generators to emulate the data collection and also a pseudorandom generator to emulate the false data injection behaviour. [Yi Huang et al., 2011] uses for their system which is in a normal state (not attacked) a Bayesian model of the random state variables with a Gaussian distribution. They changed the distribution for the malicious data. The data used by [Chaojun et al., 2015] are based on the data from the New York independent system operator (NYISO) from 2012, and generate the state data following a procedure. The attacked data are numerical, and they apply a modification of 90%, 95%, 100%, 105%, and 110% of the original numerical values.

The main flaw in these methods is usually the loss of correlation with reality. The use of non-reality-based data and arbitrarily designed attacks results in the loss of specific behaviours related to the system and attacker.

The closest but not IoT related work is made by [Cretin et al., 2018]; they developed a DSL-based testing frameworks to perform FDIA on ATC systems, and used real data from ATC to perform FDIA on them by using a DSL adapted to the specificity of the aircraft domain.

The scope of the domain on which FDIA is applied plays a big role in the way it unfolds, for example [Liu et al., 2009] relies on the number of meters and the number of state variables to find an attack vector. In IoT, the FDIA will not necessarily try to disrupt the state estimation, but rather to disrupt the data aggregation or decrease the quality and confidence in the data, to trigger actions and events.

As far as we know, there is no existing work assessing the resilience of IoT systems attacked by FDIA. Therefore, in the next section we explore different works that do not

belong to the FDIA domain, but which could bring some research avenues to our own, particularly in terms of testing methods and data generation.

3.3/ FDIA TESTING

In this section, we look at existing test methods, which can provide us with development approaches to effectively test FDIA. The aim is to position our work with test methods and to look for the appropriate one. First, we will study the Model-Based Testing (MBT) methods that can be applied to our problem through the Model-Based Security Testing (MBST) more adapted to security issues. We will then see the methods of test data generation; those two processes will allow us to study two of the main problems of FDIA testing, which are the efficient modelling of a system that can be attacked by FDIA, and the generation of realistic and effective attacks.

3.3.1/ MODEL-BASED TESTING

In this section, we briefly review work on testing methods, in particular the MBT and MBST methods. For our case study, the MBT is interesting, since it requires an effort to model the System Under Test (SUT). Modelling the system allows us to have tests closer to the reality of the field, and also to remain objective in the test cases. Above we described that one of the importance and difficulty of the work on FDIA was to be close to the reality of the systems studied, and therefore to use their real data, their real behaviours and their real attacks. The MBT methods can therefore be fully considered in our work.

The MBT domain has been studied extensively and rely on models to achieve automatic test generation [Utting et al., 2012].

The advantages of using MBT are described as follows by [Utting and Legeard, 2007]:

- The MBT is good at finding errors. Different tests have shown that the MBT always finds a greater or equal number of faults than manual testing methods. This, of course, depends on the competence of the tester and the quality of the test model.
- The MBT is cost and time efficient. Studies have shown that the use of MBT reduces costs compared to manual testing. However, some complicated models or tools may negate this advantage. In addition, the MBT saves time because tests are generated, can be easily explored and can also quickly find the causes of failure.
- The MBT has a better test quality. The use of MBT allows improving the test quality, especially thanks to a reproducible design process uncorrelated to the skills of the test engineers.

- The MBT detect defects in the requirements. By building the abstract model of the SUT, the test designer or MBT tools can detect defects. The model has precise semantics verified by the tools, which can show inconsistencies in requirements. Its design also raises many questions about the requirements and can therefore find defects.
- The MBT bring traceability. When executing the tests generated by MBT, you can always correlate the tests to the model. It allows, for example, to optimise the tests execution. If the model change on a particular requirement, the tester can generate and execute only the concerned test subset.
- The MBT accepts the evolution of requirements. With MBT, when requirements change, the test designer can easily update the test model, and generate the new tests. In a manual test, the test designer has to update all the tests, which can be very large, much larger than a test model.

The MBT has therefore a lot of interesting points. However it is very much oriented towards functional requirements testing, and is not very suitable for FDIA testing. A derivative of this system is the MBST, which is the model-based testing of security requirement [Felderer et al., 2016]. The MBST is a Dynamic Application Security Testing (DAST) who can be divided in two major categories: The Model-Based Functional Security Testing (MBFST) and the Model-Based Vulnerability Security Testing (MBVST). The first is dedicated to two aspects, on one hand, to ensure the safe operation of systems and, on the other hand, to allow the testing of security requirements, the second is dedicated to testing what is not specified in the requirements. These two sides of the MBST are widely discussed by [Utting et al., 2016]. To test the FDIA, we don't have to test security requirements, but rather to test the system resilience to attack, and therefore to test the possible vulnerabilities of the system. We are thus more interested in the MBVST methods. In their work, [Utting et al., 2016] define 3 approaches for MBVST:

1. Pattern-Based and Attack-Model-Based,
2. Model-Checking,
3. Fuzzing.

In the first one, the tester chooses not to rely on a model of the SUT but rather on a model of the attacker's behaviour. In the second one, a model of the SUT is made and a model checker is used to generate attack traces. In the third one, the method uses the well-known vulnerability testing system by injecting data, and looking if the system produces errors or crashes. This reduces the method's main flaw, which is the lack of expected results model. The two techniques that come the closest to what we want to achieve are

attack-model based and fuzzing. For FDIA testing, we try to model part of the system at the level of the data it uses, as well as its topology. This is in order to model FDIA attacks efficiently and realistically. Finally, we try to model the expected effects of attacks as well as their potential results.

We have explored the test methods that can come close to what we want to achieve. It is mainly a question of modelling the system through the attack it can undergo. We now need to explore the second point involved in performing a FDIA test, the generation of attack data.

3.3.2/ Data Generation

A FDIA aims at modifying the data. Therefore, once the attack model is made, it should be used to generate the attacked data. In this section, we will review the methods used in the literature to generate data, in particular relevant test data.

To simulate a FDIA, we need to obtain synthetic but realistic data. There are several reasons for using synthetic and generated data rather than data from real systems. The first reason is the amount of data: a system may not generate enough data to use authentic data, for multiple reasons, for example, a system that is still under development or a system that would be on confidential data. The second reason is data labelling: it is difficult to categorise authentic data, as in how to know if the recovered data are clean or altered by an attack. This brings us to the third reason: it is difficult to obtain data that are sure to be considered altered data, in particular because they were not detected as such, and even if they were, they often remain confidential. The fourth and last reason is the expressivity allowed by the use of synthetic data, rather than authentic data. Because of their rarity, authentic data labelled as corrupted means that very few use cases can be studied. The use of generated data allows for a great freedom of specification, and therefore a greater coverage of the tests performed.
It is obvious that there are shortcomings when using synthetic data. In particular, the main drawback is the realism of the data generated, and the realism of the data on the user behaviour, that is, in our case, the realism of the attacker's behaviour.

In his work, [Enderlin, 2014] presented a state of the art on test data generation. He separates it into two specific parts, the generation techniques, followed by the data generated in itself. In the next two sections, we will follow the same structure.

3.3.2.1/ Generation Techniques

A lot of work has been done in data generation for testing purposes. Many methods have been used, especially for software and web testing, and a large part of them have been

using random data generation methods. The fuzzing mentioned above is one of them [Godefroid, 2007]. It consists of generating a lot of random data and injecting it into the input of a program, in order to test it by triggering faults. Although this method is very effective in finding trivial faults, it has some disadvantages, such as execution times that can be long, or some portions of the code being complicated to cover with random data. Therefore, this method mainly allows to quickly discover defects, and to highlight the elements that need more attention from the testers. To mitigate this, improved fuzzers use guided approaches, which do not rely solely on chance, by specifying certain behaviours that the fuzzer should have.
The fuzzing is a very black box testing oriented method, but other approaches of test data generation more on the white box testing side, consists in generating tests that meet certain constraints, found through code analysis, or in a more advanced way by respecting predicates and contracts written by a tester. These constraints are then solved by solvers and used to generate test data. This type of method allows a more efficient test data generation by taking into account, and formalising, the specificities of the tested programs. Their main disadvantage is their scaling: as the programs get bigger, the complexity also increases, making the resolution more complex, costly, and time-consuming.
Many methods using these techniques have therefore been developed over time, and although they are very interesting, they are mainly associated with program testing and code testing. This strongly dissociates them from what we want to do. Moreover, these methods use contract languages and complex and advanced solvers, often dedicated to a particular domain. This makes them too complex for test generation based on attack scenarios, and therefore little usable in the case of FDIA.

Other works have been done, but more by generating realistic synthetic data used for testing purposes, rather than, as previously stated, by defining constraints to solve. In their work, [Lundin et al., 2002; Barse et al., 2003] generates synthetic data for testing fraud detection systems. They define, at first, a methodology for generating the data in 5 steps: collection of data, analysis of the collected data, user and attacker profile creation, user modelling and system modelling. In short, they retrieve the operating data of the system they want to test, and then extract operating information from the system, for example to identify users with similar behaviours. They then create profiles based on statistical distributions of events, such as users found logging in at night, whereas they usually log in during the day. Next, they model the system users with finite state machines, and finally, they model the system to produce data similar to those of the targeted system. To do this, they propose to simulate the system either in software or hardware, or a mixture of both. They at last use the generated data to test and train a fraud detection system based on a neural network.
Their approach to the idea is similar to what we want to achieve in the FDIA test. However, their fields of study make the analogy complicated. They base their work on statistics of

user behaviour found in the logs, and then model this behaviour to generate the data. In the case of FDIA, there are no particular behaviours, as an IoT device follows specific algorithms, and any deviation would immediately mean an alert. The interesting idea for our work here is the use of authentic background data, to deduce elements necessary for the generation of synthetic data.

[Popić et al., 2019] reviewed data generator techniques and use cases. For the data modelling process, they present several methods such as [Hoag and Thompson, 2007] and [Rabl and Poess, 2011]. This type of data generator describes its model through the characteristics of its data in an XML format. Then, the data generation is made from the XML model. For example, in [Anderson et al., 2014], authors present a framework for generating synthetic data for IoT. Their approach comes from the fact that it is difficult, in a big data context, to work on large amounts of data. Especially because IoT has completely different data structures between the different IoT systems. Sharing this data also brings up data confidentiality issues in corporate contexts, and therefore cannot be released to the public. Their approach proposes to generate synthetic data from authentic data, keeping the structure and characteristics contained in the authentic data. To do so, they proceed in two main steps: data characterisation and data generation. On the data characterisation step, they extract two information from the data, the structure of the XML data and the values present inside this structure. The values are classified and characterised according to several elements such as their probability or distribution. In the data generation step, they use the information harvested in the previous stage to reconstruct the data structure, and to populate it, using random generator based on the distribution of the values.
Here, the generation work is very interesting, in particular the preservation of the data structure and the characterisation of the data values from real data to deduce statistics. However, in this work, the data lose all correlation between them. For example, a temperature sensor next to a humidity sensor will have its data correlated with the previous ones but also with those of the humidity sensor. The approach proposed will probably deduce certain characteristics, such as an interval of temperature evolution, but the data will probably lose all correlation between them or the other sensor. This correlation between the data is a critical aspect in our case, a FDIA must be consistent with the data preceding or following it, as otherwise it would be detected too easily.

Finally, the method we find the most interesting is used by [Cretin et al., 2018], within the context of FDIA applied to the field of Air Traffic Control (ATC). To assess the FDIA resilience of ADS-B, an air traffic control technology, they use a DSL to model FDIA scenarios. These scenarios are then applied through a framework to actual air traffic data in order to generate altered data. Although addressing the FDIA, their DSL is very oriented towards the ATC domain. It includes language elements specific to the data present in aircraft recordings, such as altitude, ICAO aircraft model identification, or transponder code.

The use of a DSL in this kind of case studies and in highly specialised fields are interesting: this makes it possible for experts in the field to quickly get to grips with it by offering them great expressiveness. Moreover, to use a DSL it is not necessary to be familiar with general programming languages. In addition, the use of a DSL is also found in many other works for scenarios description and synthetic data generation as in Fremont et al. [2019]. The authors propose a probabilistic scenario description language for perception systems. They generate synthetic data from these scenarios, describing car positioning scenes. This is in order to train and evaluate machine learning tools, especially on rare events, or on their performance in particular conditions. DSL are very indicated to treat problems in very precise fields in an efficient way. They allow us to model and specify many aspects, including scenarios, and so are interesting for our case study, where we want to model attack scenarios and then apply them to datasets.

3.3.2.2/ GENERATED DATA

In our case study, we are interested in IoT data. Through the different developments of IoT, the data exchange format that has become preponderant is JSON. JSON is composed of two structuring elements: key/value pairs and ordered lists of values. These same values can take three forms: objects, tables and values. JSON values have six different types: strings, number, boolean, null, arrays and objects [Bray, 2017][1]. We should therefore focus on the data generation of the types listed above.
In his work, [Enderlin, 2014] considers two types of data: strings and collections/arrays.
Strings can generally be represented through either a regular expression, or a context-free grammar, which are respectively type 3 and type 2 within the Chomsky hierarchy [Chomsky, 1956]. The main problem raised to generate data using grammar-based test methods is the combinatorial explosion. Some grammar-specific elements, such as the repetition operators (+: one or more times, *: zero or more times) have no upper bounds: using data generation, huge amounts of data can then be generated. Several methods have been implemented to solve this problem. We can, for example, find methods using languages to limit the size of the repetition operators. Other methods use annotation systems, for example using tags or probabilistic weights, to help explore the grammar and limit the combinatorial explosion.
Arrays and collections are widely used in many fields. Here, for their generation, the methods combine two types of solver, one reasoning symbolically on the characteristics of the table, and the other using constraints on finite domains. The main difficulty of these methods is the communication between the solvers; they each have their own constraint systems, and their own formalisms, which makes it difficult to interface them. These methods are interesting because in order to perform FDIA, we need to generate

[1] https://www.json.org/json-en.html

very specific test data. However, these types of methods are not adapted to our use case, as indeed these methods generate data only from scratch. In our case, in order to be as close as possible to the behaviour of the data and to respect their characteristics, it is more interesting to directly modify existing data. In the case of a pure creation of FDIA from scratch, it is also necessary to find methods that allow respecting the behaviour and characteristics.

3.4/ CONCLUSION

In this chapter, we first explored the IoT by mainly considering security issues. To do so, we studied the IoT architecture layers one by one, and identified the issues related to each one. This has revealed that there are many different security issues related to each layer, and one in particular stood out from the rest, that is False Data Injection Attack (FDIA). Indeed, the alteration of data sent by IoT devices can be performed at all architectural layers, and if a layer is affected, then all layers above will be affected as well. FDIA are consequently ubiquitous within the IoT, because FDIA addresses all layers as well as the heart of the approach, which is the data. We must analyse and categorise in detail which kinds of sensors are used by IoT and the data they produce. This work will be proposed in chapter 4.
We noticed that FDIA is a very well-studied field in the case of smart grid security, and that it is also quite explored in the case of CPS and WSN. In our case of IoT, the domain has, however, been poorly studied.
After defining the FDIA, we explored the field of testing, with the aim of studying existing methods that could be applied to the challenge of FDIA testing. First of all, we explored the MBT and its subparts, which are more adapted to security testing. We concluded that the use of models was very interesting in our case, firstly to model the environment and topology of the system, and then even more so to model the attacks, for which the use of Attack-Model-Based is very appropriate.
After exploring the models, we looked at data generation. We explored existing methods for generating test data and concluded that it is more interesting to focus on modifying real data, in order to preserve their characteristics and behaviour. The generation of data from scratch is a minor part of the approach, dedicated to address a very specific trigger.

Based on the state of the art, we focused on the development of a DSL, as it allows to deal with specific and complex domains in an efficient way, providing a level of abstraction of the domain. The use of a DSL provides domain experts with a dedicated tool that does not require knowledge of general programming languages, and that uses the jargon of the domain to be addressed. FDIA is a complex type of attack that requires a good under-

standing to be treated efficiently, and the domains targeted by FDIA are often specialised and use specific jargon: the use of a DSL is therefore quite appropriate to consider these expertise needs. Moreover, to deal with security attacks in general and thus FDIA, it is interesting to define scenarios explaining the different actions that the attacker must perform to carry out his attack. DSL are very suitable for this purpose, and so we propose a method using a DSL to model FDIA scenarios. These elements are developed in chapter 5.

To begin our contribution, the next chapter focuses on analysing the data that exists within the IoT systems. This then allows us to understand the characteristics and behaviours of the data that have been mentioned throughout this state of the art, and therefore how to perform FDIA on this type of data while preserving their essence.

II

CONTRIBUTION

4

Data Analysis

Contents

4.1 Sensor Classification 47
4.2 Data Classification 50
4.3 Conclusion 52

In this chapter, we focus on the data. To effectively address our issues related to the FDIA, that is the creation of an attack scenario allowing the generation of altered data and therefore the simulation of FDIA, we must explore the IoT ecosystem. This analysis is focused on the level of the perception layer, because it is the architectural layer where the data are generated. In the following, we will thus focus on the sensors that are within this layer, and provide a classification of them. We will then analyse the data gathered by these sensors and provide a classification of their characteristics. Flowbird systems include more than two hundred different sensors, ranging from simple door opening sensors to energy consumption sensors. Our analysis is mainly based on what exists within the company, in association with open IoT datasets.

4.1/ Sensor Classification

The identification of IoT data properties is based on the analysis of the entity that provides the data. In IoT, it is the sensors which provide the data to the rest of the network. Sensor classification schemes have already been defined for electronic and engineering purposes [White, 1987; Hulanicki et al., 1991]. In the most comprehensive study of these two, nine classes were defined (acoustic, biological, chemical, electric, magnetic, mechanical, optical, radiation, thermal and others) that encompass all sensor types. In [Sikder et al., 2018], authors define for the IoT three categories for sensors: environmental sensors, motion sensors and position sensors. For this work we have identified the most widely used sensors in IoT and classified them under categories as shown in the table 4.1. We have constructed this classification because the behaviour of the data will

Sensors classification	Description	Sensors example	[White, 1987] Classes
Thermal sensor	Measure the thermal property, used for monitoring the sensors environment, or the sensor's device itself. Used in a very wide variety of application	Thermometer	Thermal (Temperature)
Mechanical sensor	Measure mechanical force like pressure, load, torque or strain	Barometer	Mechanical (Pressure)
		Load cell	Electric (Potential difference)
Proximity sensor	Measure the distance or detect the proximity of an object	Inductive sensor	Magnetic field (Amplitude)
		Photo-sensor	Optical (Wave Amplitude)
Levels sensor	Measure the levels of fluids	Water level sensor	Electric (Conductivity)
Motion sensors	Measure the linear acceleration or rotation	Accelerometer	Mechanical (Acceleration)
		Gyroscope	Mechanical (angular position)
Optical sensor	Measure optical property, lots of sensors operate in the infrared	IR sensor	Optical (Spectrum)
		CMOS sensor	Optical (Wave amplitude)
Chemical sensor	Measure the chemical environment of the sensors (gas, humidity, particle)	Hygrometer	Chemical (Concentration)
		Particle sensor	Optical (Wave amplitude)
Acoustic sensor	Measure waves, as acoustic waves or ground waves	Microphone	Acoustic (Wave amplitude)
Position sensors	Measure a relative position	Compass	Magnetic field (Amplitude)

Table 4.1: Sensor classification

change depending on the sensor type; we need to get as close as possible to the actual behaviour of the data for the purpose of data alteration, mainly to avoid being easily detected by possible intrusion detection tools. This classification will help in the modelling and execution of attacks, by characterising the data that the attack will alter.

In the following, we present the nine classes we have identified in explored systems.

Thermal sensors: Thermal sensors measure the changes in temperature property of a particular element, such as body temperature, room temperature or car engine temperature. This type of sensor is used in a wide variety of applications. In the case of IPS, this type of sensor is used to monitor the state of the environment around, or inside, the parking devices.

Mechanical sensors: Mechanical sensors measure changes in mechanical forces, usually pressure, load, torque and strain. They are widely used in the industry area for the control of machinery and biomechatronics to reproduce the behaviours of the natural sensors of the human body. This type of sensor is not used in the case of IPS.

Proximity sensors: Proximity sensors are used in a very wide variety of applications, the most common one being in smartphones, for detecting the presence of something near the screen (face, pocket), in order to prevent bad input. In the case of IPS, this type of sensor is used to detect the presence of a person in front of parking devices.

Level sensors: Level sensors detect the level of a fluid, such as water but also the levels of solids, such as grain and powder. The sensor type will differ depending on the target, for example conductive sensors for water, and rotating paddle sensors for solids. This type of sensor is not used in the case of IPS.

Motion sensors: Motion sensors measure the position in space through the linear acceleration and rotation of the object monitored. These types of sensor are usually accelerometers and gyroscopes. The most common use case is in smartphones. In the case of IPS, this type of sensor is used to detect attempts of physical intrusion on devices.

Optical sensors: Optical sensors deal with light, by converting light properties in electrical signals. Lots of devices use this type of sensor for a wide range of applications. They can be found in a smartphone to adjust the screen luminosity, in a heartbeat sensor or in an infrared thermometer. In the case of IPS, this type of sensor is used to read QRcodes.

Chemical sensors: The chemical sensors measure the chemical properties, usually the concentration, of a specific element. In the case of IPS, this type of sensor is used to measure the NO_2 concentration around devices.

Acoustic sensors: Acoustic sensors convert the wave amplitude of a sound to data. The most common example of an acoustic sensor is the microphone. In the case of IPS, this type of sensor is used to measure the noise around devices.

Position sensors: The position sensors measure the positioning in space of the object they are in. The most common example of a position sensor is the compass. In the case of IPS, this type of sensor is used to detect the position of the devices.

Usually the sensors are used with two main intentions:

1. to gather information in order to perform control and operate actuators,
2. to monitor elements, either to ensure the proper functioning of the system, or to take a decision based on the monitored data.

In FDIA, a malicious adversary carries out attacks through these two main objectives. In

the case of a thermal sensor for example, an attacker can disturb the function of an asset by modifying the temperature returned by the sensor, or by generating an anomaly to trigger the maintenance of the attacked part.

One of the great challenges of the FDIA on IoT systems is that devices are generally not alone in their environment, depending on the type of system. A citywide system for monitoring the street environment (temperature, hydrometry, particles, pollution) will need several sensors to get sufficient and reliable coverage (one device per street, about a hundred metres apart). A smart car will need several sensors, close to each other, to analyse the road (multiple ones per side of the car, a few centimetres apart). Due to this usual configuration of multiple devices, the spatial propagation of the measurements must be addressed. For example, in thermal sensors the temperature propagation must be considered for the attack to remain coherent. In some cases, if two sensors are close to each other, the data from the two sensors must be altered according to their distance and the magnitude of the desired alteration; it also depends on whether the attacker tries to attack through the altering of a local property, or a more global property.

4.2/ Data Classification

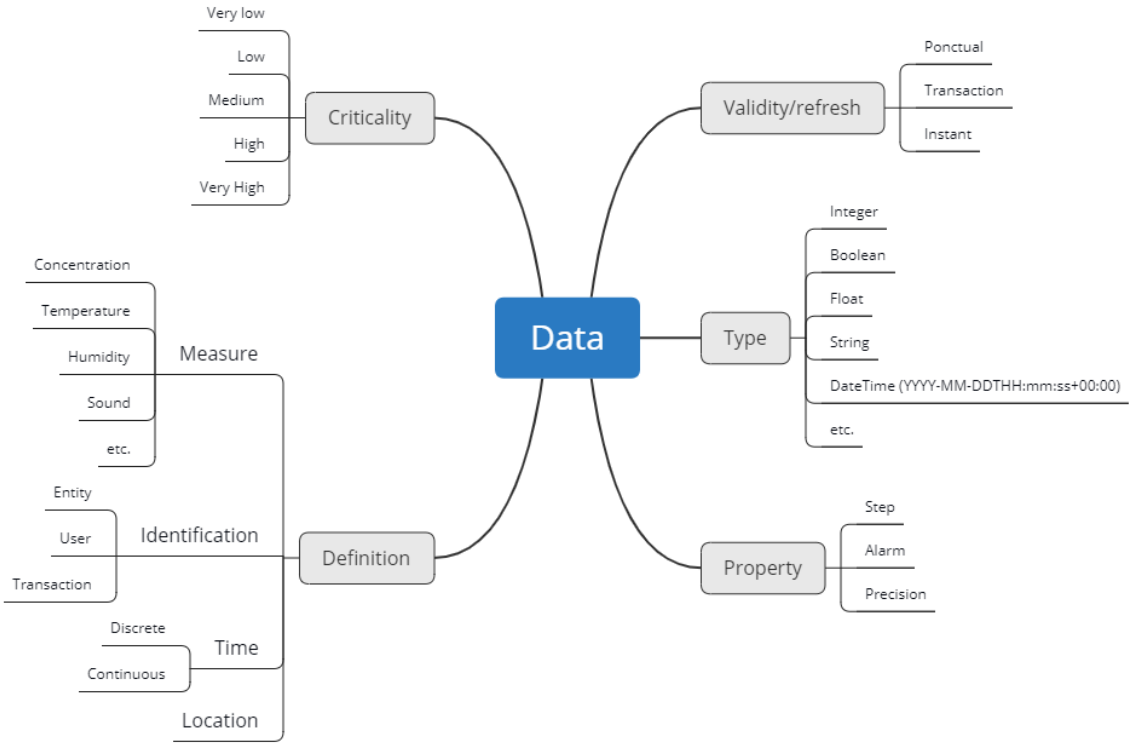


Figure 4.1: Categorisation of a data

After categorising the sensors, we need to analyse what defines the data gathered by

the sensors. In order to do so, we analyse multiple datasets of IoT systems to find which property defines data sent by an IoT sensor.

Through the research, we define that an IoT data is characterised through five properties: definition, criticality, validity, property and type. This classification is summarised in Fig. 4.1.

Definition: it is the representation of the data, it can be a data of *identification*, *time*, *location* or a *measure*.
The data of *identification* is always present in IoT datasets, this data serve to identify, either the device itself, the transaction or the user of the sensor. For FDIA, identification data are important, their main purpose being to select the target that the attacker wants to attack.
The *time* data are also always present in an IoT dataset, they are usually in a timestamp format. These data are used to situate the data over time. For FDIA, they are useful for creating scenarios within a specific time frame: the time data needs a high degree of consistency to remain undetected.
Location data represents the location across space of the device associated with that data. The location is not present in all IoT dataset. However, for effective tampering and attack propagation through the IoT system, it is important to have this data available, as it can be added by the attacker. This data should be set apart from location data that does not represent the position of the concerned devices: such data would be found in the next category.
The *measure* category contains all the data returned by the sensors described in the previous section. The decisions taken by the application layer are generally based on those data; therefore they are usually the main target of the attacker.

Criticality: it is a scale of the severity of the impact that could occur when specific data are attacked and altered by FDIA. This property is useful to get a precise enough definition of the probability of an attack on the data. Moreover, this property helps the scenario maker to build the attack by selecting the most critical data. This scale is defined through five levels: very low, low, medium, high and very high. This scale is typically used in the standard ISO/IEC 27005 for information security risk management.

Type: it is the data type. Through our analysis we found that all data types that can be found in programmatic languages are used in our data (Integer, Float, String, Char, Bool, etc.). However, the vast majority of the data rely on a JSON data format for their data transaction, and therefore, the most used types are String, Number and Boolean.

Validity: it is the data validity, through the different transactions. In IoT, the data are usually valid only during their own transaction, but occasionally some data can be sent for information or configuration purposes, and thus can be valid during several transactions.

Property: it represents the particular properties of the data. We have identified specific properties such as steps, precision or whether the possibility that the data are alarms.

All these properties can evolve in the future. They are based on the data we have analysed while keeping in mind to be as exhaustive as possible. Despite the attempts of several organisations, IoT devices are nevertheless not standardised, or only slightly so. Thus, it is difficult to map them all, and there may be specific properties that are not listed.

4.3/ CONCLUSION

In this chapter we have answered the research question RQ1. We have identified a categorisation of sensors that are the most likely to exist within IoT systems. To do this, we analysed the different systems and services present in the company, and explored some external systems as well, to confirm and complete the observations made internally. From these systems and services, we have recovered the data that passes through them to extract a categorisation of the data present within IoT systems. It is difficult to exhaustively map all the IoT sensors and their associated data existing in the world, notably because of their great heterogeneity. Nevertheless, with these two categorisations, we can put properties on the data that can be used in the different phases of the approach. Thanks to these classifications, we can directly help the experts during the modelling phase by giving them the different information they need. The modelling language also uses this information as metadata to define the elements it works with, particularly the type of data and its definition as in described in chapter 5. These metadata will also be useful in the context of improvement by machine learning methods. If we want to assist or replace the expert modeller with machine learning methods, we must assist the training by using metadata to provide information, in addition to the raw data.
We now have information on the fundamental elements of the IoT and the FDIA, the data. We must now model the attacks. This modelling through a dedicated language is explored in the next chapter.

5

Domain Specific Language for FDIA Design

Contents

5.1 Decision **54**
5.2 Domain Analysis **56**
5.2.1 Scenario 57
5.2.2 Scenario Properties 57
5.2.3 Scenario Actions 58
5.2.4 Selection Criteria 58
5.2.5 Alteration Criteria 59
5.2.6 Time Frame 59
5.3 Design **59**
5.3.1 Scenario 60
5.3.2 Scenario Actions 61
5.3.3 Selection Criteria 62
5.3.4 Alteration Criteria 63
5.3.5 Time Frame 64
5.3.6 Grammar Utility 64
5.4 Implementation **65**
5.4.1 Language Implementation 65
5.4.2 Language Interpretation 67
5.5 Conclusion **67**

This chapter is dedicated to the design of the DSL, based on a methodology that allows us to explain why we choose to use a DSL, and how we develop it. To do this, we first study the decision to use a DSL, then we analyse the application domain, followed by its design and finally its implementation.

We use the work proposed by [Mernik et al., 2005] who describes Domain Specific Language (DSL) as languages tailored to a particular domain and application.

Definition 4: Domain-Specific Language

"A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain." [Deursen et al., 2000]

They have more expressive power than General Programming Language (GPL) such as C or Java, and have many advantages. Due to their ease of use compared to GPL, they allow gains in productivity as well as in maintenance costs. In particular, they allow a gain in productivity for users who are experts in the field in which the DSL operates. It also allows users who are not fluent in GPL to learn and perform complex actions on the specific domain. Indeed, in general, DSL provide a language that is often close to the natural language used by experts in the field, making it quick and efficient to learn. We can find in the history many DSL that have shown their usefulness in addressing very specific areas with efficiency, performance and simplicity, such as SQL for databases, CSS for stylistic description of web pages, or shell script for filesystem manipulation.

Of course, DSL also have several drawbacks. Firstly, as their name implies, they are dedicated to a domain, and so they are more complicated to learn than traditional languages, especially when it comes to documentation and code examples. Secondly, building a DSL from scratch to address a domain is very expensive, whether it be in terms of implementation, support, maintenance or user training. The choice to develop a DSL must therefore be analysed to determine whether it is an appropriate solution. For this purpose, [Mernik et al., 2005] in their article "When and How to develop DSL", describe a multi-step methodology to ensure that a DSL is the right answer to the problem, and how it should be developed. They define four steps: decision, analysis, design, and implementation. The decision step is the answer to "When to develop DSL" and the other steps are the answers to "How to develop DSL". In the following sections, we review the definitions of each step proposed by the authors, and give a motivated description of our DSL using this methodology.

5.1/ DECISION

As noted above, the decision to use a DSL is not a trivial one, neither is the decision to use an existing modelling language (textual or graphical), or to build one from scratch. To aid in this decision process, [Mernik et al., 2005] defined nine decision patterns, such as task automation or GUI construction. In the following, to evaluate the choice of a DSL, we describe the constraints associated with our context and then analyse it with the decision patterns provided to motivate our DSL.

As mentioned previously, a FDIA on IoT systems is an attack that will alter data sent by one or more devices. These devices can be very numerous and can send a very large amount of data in a very short period of time. To be able to test the resilience of the systems under attack, we need to simulate these attacks. To do this, we can either decide to build an attack from scratch, taking the data structures to be attacked, producing corresponding data, and then injecting them into the attacked system, or we can take original datasets from IoT devices, modify them to create attacks, and then inject them into the attacked systems.

Our DSL can be considered as scenario-centric. It aims at describing the scenario of FDIA; a scenario is composed of several elements:

- A data record from a real IoT system. The FDIA are applied to this record to generate a new, altered record.
- The scenario configuration where the data structure and the data property are described. It is the description of the record.
- The scenario actions where the steps to model FDIA are described. These will then be applied to the record to generate the attack.

In the case of a really simple FDIA, such as sensor disruption, where an attacker simulates an out-of-service sensor by sending the same data over and over again, it would be simple and resource efficient to create the attack by hand, by replacing all the values by another one in the recording file, or by using a more automated script, such as a replace/search function on a text editor, or in a basic scripting language. However, in the case of a much more complex attack, such as a gradual increase of a value over a geographical area, triggered when a threshold of another value is exceeded, it would be extremely complicated, time consuming and difficult to modify the recording by hand, or time consuming and expensive to develop an adhoc script specific to this attack scenario. Hence, the need for a tool that allows to simply describe and execute a complex attack scenario, composed of many alterations, with many different properties.

Considering all of this, several decision patterns are highlighted, allowing us to consider that according to these patterns, the use of a DSL in our case study would be quite appropriate. The highlighted patterns are *notation*, *task automation* and *system front-end*.

- Notation: this pattern is divided in two parts, either transforming a visual notation into a textual notation, or making a user-friendly notation for an API. In our case, this is the user-friendly part. Indeed, we can consider the language used by experts discussing FDIA as an oral API between them. Experts discussing FDIA in very

specific areas will also have specific ways of presenting them. In this case, using a language close to the one used by expert users, allows us to describe test cases and attack scenarios in an efficient way.

- Task automation: this pattern is described for a language that simplifies and automates the tasks performed on a GPL, which are long, tedious and repetitive. In our case, those patterns are really present. As said above, modelling FDIA can be a tedious task in terms of time and complexity. A DSL would therefore allow its automation.
- System front-end: this pattern is similar to the first one. It describes the fact that a DSL proves useful in the case where it is used for a better handling of system configuration and adaptation by the front-end users. In our case, this can be seen as the first pattern, which aims to provide expert users with a front-end interface for modelling attacks, both in configuration and substance.

The decision to make a DSL is thus quite appropriate according to these patterns. The creation of a DSL will make the experts work easier both in terms of tediousness and complexity. So, we answered the *"when to develop DSL"* proposed by [Mernik et al., 2005]. The next step in the methodology is therefore the analysis of the DSL domain: we move on to the question *"how to develop DSL"*.

5.2/ Domain Analysis

In [Mernik et al., 2005], the authors define the analysis part as being the part in which one seeks to clearly identify the field, in terms of its problems and its level of knowledge. Several sources of input can be used for this, including internal and external knowledge. These may include technical documentation, research by experts in the field, existing GPL code, or user and customer surveys. The output can also be varied: generally, it relates to a domain-specific terminology with its semantics, and both formal and informal analysis are used.

In the conducted analysis to define a DSL, are highlighted three recurring patterns: these are the formal, code-based and informal analysis. Most of the domain analysis work to develop a DSL is done informally. Nevertheless, a few methodological approaches were proposed a long time ago, such as DARE (Domain Analysis and Reuse Environment) [Frakes et al., 1998] or FODA (Feature-Oriented Domain Analysis) [Kang et al., 1990].

The output of a formal domain analysis methodology is a domain model consisting in:

- a domain definition,

- a domain terminology,
- a description of domain concepts,
- a feature model of the domain concept commonalities and variabilities.

These methods have obtained good results in terms of domain analysis. However, in addition to being now relatively old, they have proven to be complex in terms of resources and effort to implement. In particular, the lack of guidance and guidelines can lead to errors in the supposedly formal analysis. A more recent formal method is also used for the domain analysis, namely ontologies [Tairas et al., 2009]. However, according to [Kosar et al., 2016], the vast majority of papers describing DSL use an informal method for domain analysis. In their study, they showed that 93.3% of the selected papers that did a domain analysis used an informal method to describe it. Our domain, which is IoT, is technically a high level and very heterogeneous domain, taking place in a highly industrialised universe. It would therefore be complicated to formally define the entire domain encompassing it. In the following, we therefore carry out the domain analysis of our DSL informally.

5.2.1/ SCENARIO

The central figure of our domain is the notion of scenario. A cybersecurity scenario is a complete step-by-step description of the actions taken by the attacker to carry out an attack. In our case we want to express the steps to simulate a FDIA. For this, two main elements are required: the global information of the scenario, that we call the **scenario properties**, and the step-by-step actions of the scenario, that we call the **scenario actions**. A scenario is also linked to a specific data record, to which it must be applied.

5.2.2/ SCENARIO PROPERTIES

Scenario properties are the elements that provide information to be used globally in the scenario. These properties must be scalable. According to the needs of the scenario actions, new properties not thought of in the design should be easily added. This also brings the notion of mandatory property: some properties such as the scenario naming must be mandatory, while the presence of some other properties, such as the use of geolocation, may be entirely dependent on the scenario actions.

5.2.3/ Scenario Actions

Scenario actions are an ordered list of actions performed by the scenario to execute a FDIA. When interpreting them, they should be read and interpreted in the order in which they appear. This implies an incidence of previous actions. We propose to define an action using an attribute. We identify four basic attributes for performing a scenario action. In the following, we refer to these attributes as alteration primitives. They allow to define the general action that will be performed by the scenario action. They also define the elements that compose the action. The four proposed alteration primitives are as follows:

- **Create** is used to generate data from scratch, using the information provided in the action. The generated data are inserted into the selected record.
- **Alter** is the action used to modify the data in a record, using a selection criterion to select the data to be modified, and an alteration criterion for the modifications to be applied to the selection.
- **Copy** is the action used to copy data from a record selected by a selection criterion. The copied data are then altered, using an alteration criterion, and are inserted into the original record.
- **Delete** is the action used to delete data from a record selected by a selection criterion.

These four basic primitives can be extended later to add more complex and comprehensive primitives. Indeed, some specific subdomains of the IoT may require alterations with specific parameters.

Following the primitives, the action is composed of multiple parts defined by the alteration primitives. In the following, we explain the main parts that compose a scenario action: the *selection criteria*, the *alteration criteria*, and the *time frame* associated.

5.2.4/ Selection Criteria

Selection criteria are used to select the specific records that the attack designer wants to alter, it is the target of the action. Usually, the selection is made through the identifier of a device, but it can also be a specific condition such as a specific value going above a threshold, and in this case the selection can be seen as a trigger.

Moreover, the selection criteria must be scalable, to suit the different types of data and IoT devices that may be present in the records. A specific selection can be for example the selection inside a circle, defined by geographical coordinates for its centre, and a distance in meters for its radius.

5.2.5/ Alteration Criteria

Alteration criteria are the effects of the action. These effects are applied to the specific records previously selected by the selection criteria. The most fundamental criterion for alteration is a simple assignment of value; the value can range from a simple integer to a complex function. Nevertheless, all alteration criteria are derivatives of a value assignment: for example, value assignments, but in increments over time. Like the selection criteria, the alteration criteria must also be scalable, to deal with the different elements that may exist in the records.

5.2.6/ Time Frame

The time frame is the notion used to define the temporality of the associated action. It can be represented in two different ways: either absolute, where we consider the first message of the recording as the point of origin of our timeline, or in a relative way, by directly using the notion of time present in the recordings. This property can also be seen as a specific selection criterion, and could have been part of the selection criteria. However, the notion of time frame is very important in the realisation of FDIA, and it is therefore important that they appear, in a mandatory way, in the action definition.

In the next section, we explore the third step of the methodology, the design of the DSL.

5.3/ Design

The design step is the one that allows us to technically define the language. This phase defines the semantics and syntax of the language using the information that has been collected during the domain analysis. Several patterns have been defined within the same methodology for the design of a DSL, either based on existing languages (*"Language exploitation"*), or by inventing a language and his grammar from scratch (*"Language invention"*). To do this, we have to analyse existing languages that could be reused, at least partially, to facilitate the design of the DSL. When we find an interesting language for our case study, three recurring patterns are possible: either use the language partially by retrieving parts of the grammar (*"Piggyback"*), restrict the language by removing parts of the grammar (*"Specialization"*), or extend the language by adding rules to the grammar (*"Extension"*). It is much easier to develop a DSL using existing languages than to start from scratch. Reusing existing languages also allows faster learning for people who already know such languages. In our DSL, multiple aspects of the language can be seen as *language exploitation*, such as the general formatting of actions which is similar to a query language, or the use of an infix operator notation for arithmetic operations.

In the domain analysis section (see section 5.2), the domain was cut in multiple parts. These parts can be seen in the grammar as nonterminal lexical elements. In the next subsections, we explore the design of the DSL through the different parts defined on the domain analysis. The different following presented grammars are in the Extended Backus-Naur Form (EBNF) [ISO/IEC, 1996].

We provide in Fig. 5.1, a short example to illustrate a complete FDIA scenario and explain each DSL part of it. It is not an actual scenario, but it allows us to illustrate the design.

```
scenario "failsensor"                                                          } Scenario properties
ticker 2
geolocation (47.239195, 6.022537)
create things set humidityRH = (0 -> 100,2) from 0 to 4500;                    }
alter things where meter_code = "10" set temperatureTC = 0 from 0 to 4500;     } Scenario actions
copy things where meter_code = "10" set meter_code = "12" from 0 to 10000;     }
delete things where meter_code = "12" from 50 to 600;                          }
```

Figure 5.1: Example of a FDIA scenario

5.3.1/ SCENARIO

The scenario, as stated previously, is the main part of our DSL. It represents the starting symbol of the grammar, and aims at describing the complete action of a FDIA. A scenario consists of a set of properties and a specific combination of actions, in order to create a FDIA. It can therefore be divided into two parts, the scenario properties and the scenario actions.

⟨*scenario*⟩	::=	⟨*scenarioDeclaration*⟩ ⟨*executionList*⟩
⟨*scenarioDeclaration*⟩	::=	Scenario ⟨*string_literal*⟩ ‘;’ (⟨*scenarioProperties*⟩ ‘;’)*
⟨*scenarioProperties*⟩	::=	ticker ⟨*decimal_literal*⟩
	|	geolocation ‘(’ ⟨*real_literal*⟩ ‘,’ ⟨*real_literal*⟩ ‘)’
⟨*executionList*⟩	::=	⟨*execution*⟩ ‘;’ (⟨*execution*⟩ ‘;’)*

Figure 5.2: EBNF Grammar of a scenario

In the EBNF grammar in Fig. 5.2, this is represented by the two nonterminal `<scenarioDeclaration>` and `<executionList>` derived from the production rule `<scenario>`. The rules `<scenarioDeclaration>` consists in the information applied to the entire scenario and are useful for its execution. The name of the scenario is the only mandatory property, and at this moment we have defined two optional properties:

1. *ticker*, which is an indication of the elapsed time between the sending of two messages by the SUT. Usually, IoT systems send their data at a regular rate, so we

need to implement the ticker property, to avoid being detected by potential detection systems, when we inject a new record.

2. *geolocation*, which is a property used to locate in space the point of application of the scenario. In our context, we only use a two-dimensional location: latitude and longitude.

These properties can be extended by adding alternatives production rules to the nonterminal <scenarioProperties> of the grammar. For example, in the case of the addition of a property allowing three-dimensional geolocation, we must add a <real_literal> to consider altitude. We assume that in our context, we only need to use a 2D representation.

An example of a scenario start, without the list of actions to perform, would be:

```
1 scenario "exampleScenario" ;
2 ticker 2 ;
3 geolocation (47.237829,6.0240539) ;
```

Following the scenario properties, we present the rule <executionList> which is the list of actions that the scenario uses to model the FDIA.

5.3.2/ Scenario Actions

Scenario actions are the core of the attack execution. It is a step-by-step description of the actions that the scenario must perform.

⟨*execution*⟩	::=	⟨*create*⟩
	|	⟨*alter*⟩
	|	⟨*delete*⟩
	|	⟨*copy*⟩
⟨*create*⟩	::=	create things ⟨*alterationCriteria*⟩ ⟨*timeframe*⟩
⟨*alter*⟩	::=	alter things ⟨*selectionCriteria*⟩ ⟨*alterationCriteria*⟩ ⟨*timeframe*⟩
⟨*delete*⟩	::=	delete things ⟨*selectionCriteria*⟩ ⟨*timeframe*⟩
⟨*copy*⟩	::=	copy things ⟨*selectionCriteria*⟩ ⟨*alterationCriteria*⟩ ⟨*timeframe*⟩

Figure 5.3: EBNF Grammar of the scenario actions

An action is composed of several parts, that can be seen in the grammar in Fig. 5.3. The first part is the nature of the action, we call it the alteration **primitive**, and the following parts of the action are defined by this primitive. As indicated in the domain analysis, we have three recurring parts needed to perform a FDIA: the part where the records are selected to be impacted by the related action with the rule <selectionCriteria>, the

part the alterations to be made on the previously selected records are indicated with the rule <alterationCriteria>, and the last part where the time window on which the action is applied, is indicated with the rule <timeframe>.

This is the skeleton of a scenario action:

alter things where identifier = ”10” set temperatureTC = 0 from 0 to 4500;

Primitive — Selection criteria — Alteration criteria — Time frame

In the next sections, we explore the two criteria and the time frame.

5.3.3/ SELECTION CRITERIA

Selection criteria allow specific messages in the record to be selected according to a query.

⟨*selectionCriteria*⟩ ::= where ⟨*selectionCriterion*⟩ (and ⟨*selectionCriterion*⟩)*

⟨*selectionCriterion*⟩ ::= ⟨*id*⟩ ‘=’ ⟨*type*⟩
| ⟨*id*⟩ ‘>’ ⟨*type*⟩
| ⟨*id*⟩ ‘<’ ⟨*type*⟩
| ⟨*id*⟩ ‘!=’ ⟨*type*⟩
| ⟨*id*⟩ isInsideCircle ‘(’ ⟨*real_literal*⟩ ‘,’ ⟨*real_literal*⟩ ‘,’ ⟨*decimal_literal*⟩ ‘)’
| ⟨*user_function*⟩

Figure 5.4: EBNF Grammar of the selection criteria

According to the rule <selectionCriteria> in the corresponding grammar in Fig. 5.4, the criteria set always begins by the terminal symbol "where", and is composed of at least one criterion; if several criteria are present, they are separated by the terminal symbol "and". This symbol is self-explanatory as it represents the logical conjunction between the different selection criteria. The logical disjunction between the criteria, with the "or" operator, is not yet implemented in the grammar. It is thus necessary to add an action performing the same alteration, using the distributivity of the operators. However, since the actions are performed step-by-step, in the order in which they appear, issues with consistency may arise in the alteration. The implementation of the disjunction will have to be done in the future: as in practice, the disjunction is rarely needed when selecting messages to be altered, we didn’t immediately see the necessity of it.

At that time, we have four basic selection criteria defined, as well as a complex one: they are the equality, superiority, inferiority and difference operators, and the complex one is a criterion that selects the messages within a circle defined by longitude and latitude, and a radius in meters. These basic selections must be able to be enriched according

to the SUT, and the specific needs of the attacks. The nonterminal <user_function> represents this possibility; it is not actually part of the grammar, it rather signifies the importance of predicting the evolution of the types of selections.

Two different examples of selection criteria untied from an action would be:

```
1 where identifier=42 and temperature > 451;
2 where location isInsideCircle(47.237829,6.0240539,500.0)
```

Following these selection criteria, we present the alterations that can be applied to the selected elements.

5.3.4/ ALTERATION CRITERIA

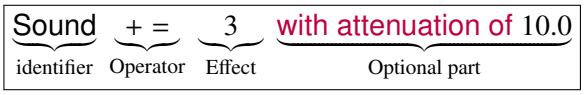
The scenario designer specifies the attacks that should be performed during the linked action with the alteration criteria. The described alterations are applied to the messages selected with the selection criteria part.

⟨*alterationCriteria*⟩	::=	set ⟨*alterationCriterion*⟩ (and ⟨*alterationCriterion*⟩)*
⟨*alterationCriterion*⟩	::=	⟨*id*⟩ '=' ⟨*type*⟩
		| ⟨*id*⟩ '=' ⟨*evol*⟩
		| ⟨*id*⟩ '+=' ⟨*real_literal*⟩
		| ⟨*id*⟩ '*=' ⟨*real_literal*⟩
		| ⟨*id*⟩ '+=' ⟨*evol*⟩ ⟨*attenuationCriteria*⟩?
		| ⟨*array*⟩ '+=' ⟨*real_literal*⟩
		| ⟨*user_function*⟩
⟨*evol*⟩	::=	'(' ⟨*decimal_literal*⟩ '->' ⟨*decimal_literal*⟩ ',' ⟨*decimal_literal*⟩ ')'
⟨*attenuationCriteria*⟩	::=	with attenuation of ⟨*real_literal*⟩

Figure 5.5: EBNF Grammar of the alteration criteria

According to the rule <alterationCriteria> in the corresponding grammar in Fig. 5.5, the criteria set always begins by the terminal symbol "set", and is composed of at least one criterion; if several criteria are present, they are separated by the terminal symbol "and". At that time, we defined several types of alteration such as the simple affectation, the basic incrementation, or the multiplicative incrementation. These alterations are always made up of three parts: the first is the identifier of the property to be altered, the second is a terminal symbol known as the operator to identify the type of alteration, and the third is the effect of the alteration. An optional fourth part can exist, used to complete the alteration of specific attacks: for example in our grammar, the rule <attenuationCriteria>, which allows reducing the magnitude of the attack based on the distance of the altered object from the application point of the scenario.

This is the skeleton of an alteration:



Obviously, as in the previous sections, the scalability of the language is important. Alteration criteria must be able to be enriched with other specific alteration criterion, according to the SUT, and the needs of the attacks. This is represented by the rule <user_function>.

An example of alteration criteria, untied from its action, would be:

```
1 set temperature=42 and humidity+=(0.0->451.0,10.0) with attenuation of 10.0
```

Following the two criteria parts, we present the time frame selection.

5.3.5/ Time Frame

The time frame is part of the action, and allows it to be situated in time.

⟨*timeframe*⟩	::=	from ⟨*decimal_literal*⟩ to ⟨*decimal_literal*⟩

Figure 5.6: EBNF Grammar of the time frame

According to the rule <timeframe> in the corresponding grammar in Fig. 5.6, the time frame always begins with the terminal symbol "from", followed by the start time, and ends with the terminal symbol "to", itself followed by the end time.

5.3.6/ Grammar Utility

In order not to overload the other sections with superfluous elements, all the grammar rules have not been presented. This section is therefore devoted to grammar elements which are useful for our language, but didn't fit in the previous sections: these are, in particular, the data types. The grammar is shown in Fig. 5.7

In this section, we have explored the design of the DSL through the different important parts defined in the language. The complete EBNF grammar of this DSL is shown in Fig. 5.8. In the next section, we explore the fourth step of the DSL development methodology, the implementation.

⟨*type*⟩	::=	⟨*id*⟩
		| ⟨*string_literal*⟩
		| ⟨*decimal_literal*⟩
		| ⟨*real_literal*⟩
⟨*array*⟩	::=	array '(' ⟨*id*⟩ ',' ⟨*decimal_literal*⟩ ',' ⟨*decimal_literal*⟩ ')'
⟨*decimal_literal*⟩	::=	('0'-'9')$^+$
⟨*id*⟩	::=	('a'-'z' 'A'-'Z') ('_' 'a'-'z' 'A'-'Z' '0'-'9')*
⟨*string_literal*⟩	::=	`'' ('_' 'a'-'z' 'A'-'Z' '0'-'9')$^*$ ''`
⟨*real_literal*⟩	::=	('0'-'9')$^+$ '.' ('0'-'9')$^+$

Figure 5.7: Grammar of utility elements

5.4/ IMPLEMENTATION

The implementation is the development process of the DSL, it is based on the grammar of the language, which has been defined in the previous section. The task at hand is now to make it executable. In order to do this, we make choices to render the DSL executable with respect to the six proposed patterns. The two best known patterns are not only dedicated to the DSL, but to all existing languages: they are the interpreter and the compiler/application generator. Other, more DSL-oriented, patterns exist, such as the preprocessor or embedding ones. The first is an implementation that transforms the DSL into another executable base language, and the second is an implementation that is included within a GPL host language, such as an application library.

In our case, we chose to use the interpreter implementation. The main reason for this choice is the possibility to allow our language to be as efficient as possible, by keeping the expert knowledge in the syntax and semantics. It is also easier with this to build good error reporting. Indeed, not relying on an underlying implementation allows us to thoroughly elaborate the design and syntax of the language. This allows us to really stay in the specific domain, as the name "DSL" implies. Moreover, it allows us to get rid of existing error reporting systems, and to implement a language-specific one, for example by creating a semantic analyser that checks the typing of the variables.

In the following sections, we explore the implementation that leads to our DSL.

5.4.1/ LANGUAGE IMPLEMENTATION

Our language has been developed in Java using the parser generator ANTLR[1]. This generator is widely used to create languages and tools that require a parser. From the

[1] ANother Tool for Language Recognition: https://www.antlr.org/

⟨*scenario*⟩ ::= ⟨*scenarioDeclaration*⟩ ⟨*executionList*⟩

⟨*scenarioDeclaration*⟩ ::= Scenario ⟨*string_literal*⟩ ticker ⟨*decimal_literal*⟩ geolocation '(' ⟨*real_literal*⟩ ',' ⟨*real_literal*⟩ ')'

⟨*executionList*⟩ ::= ⟨*execution*⟩ ';' (⟨*execution*⟩ ';')*

⟨*execution*⟩ ::= create things ⟨*alterationCriteria*⟩ ⟨*timeframe*⟩
| alter things ⟨*selectionCriteria*⟩ ⟨*alterationCriteria*⟩ ⟨*timeframe*⟩
| delete things ⟨*selectionCriteria*⟩ ⟨*timeframe*⟩
| copy things ⟨*selectionCriteria*⟩ ⟨*alterationCriteria*⟩ ⟨*timeframe*⟩

⟨*selectionCriteria*⟩ ::= where ⟨*selectionCriterion*⟩ (and ⟨*selectionCriterion*⟩)*

⟨*selectionCriterion*⟩ ::= ⟨*id*⟩ '=' ⟨*type*⟩
| ⟨*id*⟩ '>' ⟨*type*⟩
| ⟨*id*⟩ '<' ⟨*type*⟩
| ⟨*id*⟩ '!=' ⟨*type*⟩
| ⟨*id*⟩ isInsideCircle '(' ⟨*real_literal*⟩ ',' ⟨*real_literal*⟩ ',' ⟨*decimal_literal*⟩ ')'
| ⟨*user_function*⟩

⟨*timeframe*⟩ ::= from ⟨*decimal_literal*⟩ to ⟨*decimal_literal*⟩

⟨*attenuationCriteria*⟩ ::= with attenuation of ⟨*real_literal*⟩

⟨*alterationCriteria*⟩ ::= set ⟨*alterationCriterion*⟩ (and ⟨*alterationCriterion*⟩)*

⟨*alterationCriterion*⟩ ::= ⟨*id*⟩ '=' ⟨*type*⟩
| ⟨*id*⟩ '=' ⟨*evol*⟩
| ⟨*id*⟩ '+=' ⟨*real_literal*⟩
| ⟨*id*⟩ '*=' ⟨*real_literal*⟩
| ⟨*id*⟩ '+=' ⟨*evol*⟩ ⟨*attenuationCriteria*⟩?
| ⟨*array*⟩ '+=' ⟨*real_literal*⟩
| ⟨*user_function*⟩

⟨*evol*⟩ ::= '(' ⟨*decimal_literal*⟩ '->' ⟨*decimal_literal*⟩ ',' ⟨*decimal_literal*⟩ ')'

⟨*array*⟩ ::= array '(' ⟨*id*⟩ ',' ⟨*decimal_literal*⟩ ',' ⟨*decimal_literal*⟩ ')'

⟨*type*⟩ ::= ⟨*id*⟩
| ⟨*string_literal*⟩
| ⟨*decimal_literal*⟩
| ⟨*real_literal*⟩

⟨*decimal_literal*⟩ ::= ('0'-'9')+

⟨*id*⟩ ::= ('a'-'z' 'A'-'Z') ('_' 'a'-'z' 'A'-'Z' '0'-'9')*

⟨*string_literal*⟩ ::= '"' ('_' 'a'-'z' 'A'-'Z' '0'-'9')* '"'

⟨*real_literal*⟩ ::= ('0'-'9')+ '.' ('0'-'9')+

Figure 5.8: Complete grammar of the FD2IOT language

grammar, it produces a parser that can be used to explore the Abstract Syntax Tree (AST). The complete implemented grammar, in the format read by ANTLR, is available in appendix A. Once this grammar is compiled by ANTLR, we obtain a tree parser, with several possibilities to implement tree parsing: the visitor and the listener mechanism. The next step is to provide it with the corresponding language input, so that the parser

can interpret it.

5.4.2/ LANGUAGE INTERPRETATION

For linguistic interpretation and tree analysis, we chose to use the visitor design pattern. The visitor allows a more complete exploration of the AST in regards of the semantics, resulting in, for example, the omission of specific nodes, or the ability to loop a defined number of times.

Our interpreter was designed to parse the tree in order to build an object representation of the input language. It is then this representation of objects in memory that is executed to perform the FDIA simulation. The UML class diagram of these objects is shown in Fig. 5.9. All the rules presented in the design section (see section 5.3) can be found in this class diagram, either as a class or as an attribute. The class `scenario` is the root class and has several attributes: the previously defined scenario properties such as the scenario name, and the execution list (`execs`) which are the list of scenario actions. The class `execution` then represents one scenario action defined in the DSL scenario. In the diagram, we can see the four elements that compose an action as previously defined, i.e., the primitive, the selection criteria, the alteration criteria, and the time frame, along with their respective classes. All these classes are composed of attributes, that represent the parts making up the different nonterminals described in the previous section. The scenario described with the DSL is then fully represented with memory objects. This has several advantages, the first one being the ease with which memory objects can be manipulated through the code before execution, rather than directly during interpretation at the DSL level, allowing for better code scalability. The second one is related to the underlying tools: this allows them to build and implement scenarios graphically.

5.5/ CONCLUSION

This chapter has presented the DSL developed to address the FDIA challenge.

We have covered the creation of the DSLby using a methodology which defines ”when” we need to create/use a DSL, and ”how” to create the DSL, through four steps. Firstly, we took the decision to create a DSL. Secondly, we did the domain analysis of the FDIA challenges in our field, to extract the important information that will be used for the design step. Thirdly, we designed the DSL, which allowed us to define its technical aspect, based on the domain analysis, and thus obtaining the language grammar. In the fourth and final step, we presented the technical DSL implementation, in terms of the modelling tools and methods used to interpret the language.

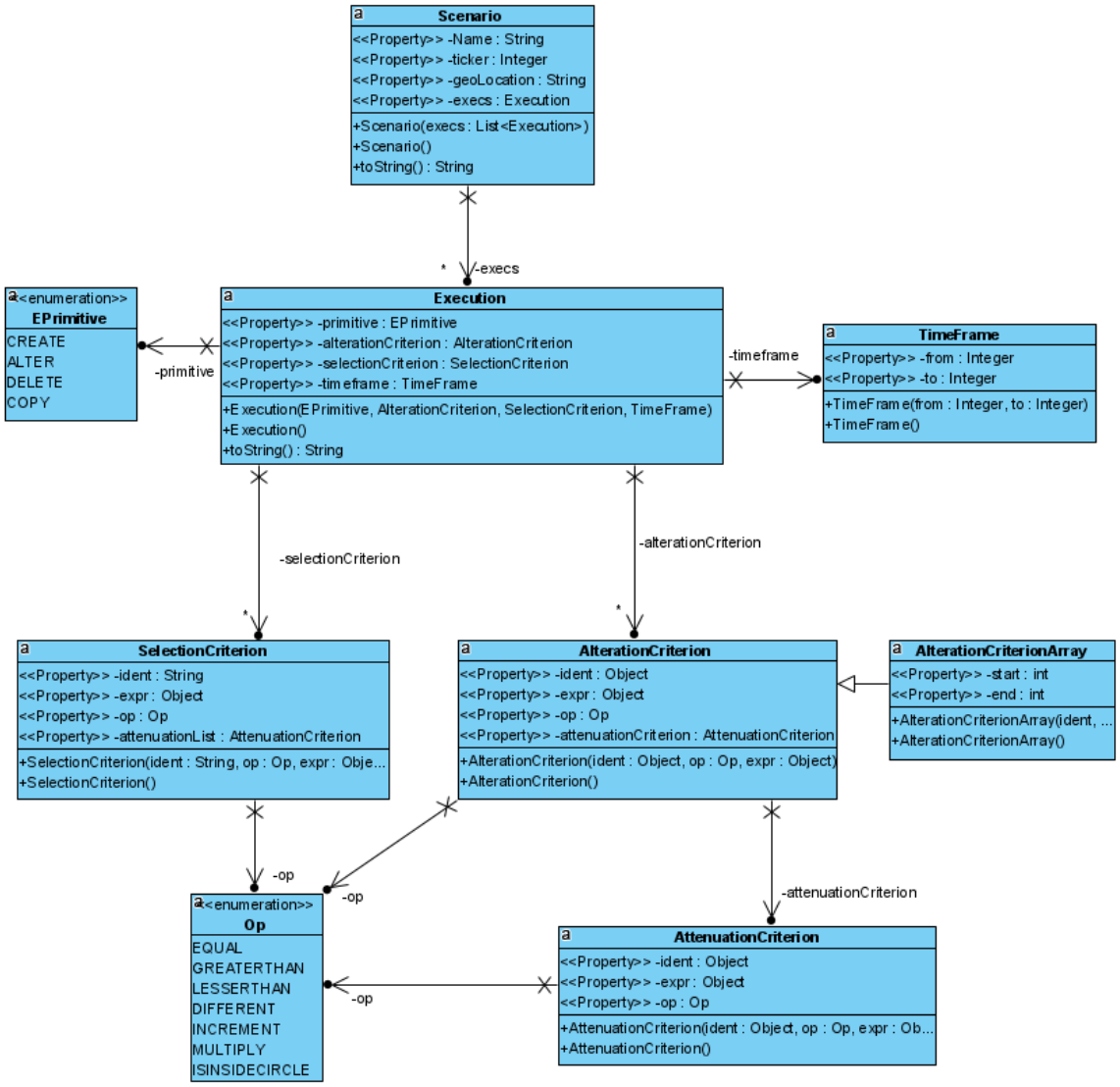


Figure 5.9: UML class diagram of the implementation in java of the interpretation

This chapter answers the research question RQ2. We have developed a language that allows us to model FDIA attacks specifically targeting IoT systems, and to be as generic as possible, with respect to the underlying domain in which the IoT system evolves.

This genericity is also addressed in the next chapter, which explores the use of DSL in general, notably through the confidence that can be placed in our DSL in terms of expressive power, quality, and in the management of the underlying tools.

6

FDIA FRAMEWORK

Contents

6.1 Workflow of our Approach 69
6.2 DSL Usage 71
6.2.1 Data Management 71
6.2.2 Validation of Alteration Primitive 73
6.2.3 Coverage of the Knowledge 74
6.3 Conclusion 79

This chapter is dedicated to the framework we have set up to generate data that are used to simulate FDIA. It focuses mainly on the DSL and the management of the data it handles. This chapter is divided into three parts: the global approach of this work, the DSL usage, and a conclusion. Concerning the usage, we discuss the use of the DSL, and in particular the level of trust that can be placed in it. We will first look at the data management used in our DSL, followed by the validation of its implementation, and finally we will explore both the expressive power and the domain coverage offered by the language.

6.1/ WORKFLOW OF OUR APPROACH

The aim of this work is to provide an efficient testing tool to assess the resilience of the IoT systems against FDIA. One of the major difficulties of this study is the total heterogeneity of IoT ecosystems. Few standards bodies have attempted to provide standard frameworks for IoT, such as oneM2M, NGSI-LD, or IoTivity, but in reality the vast majority of IoT manufacturers follow their own protocols and data representations, such as Google weave, AWS IoT Core, or Azure IoT hub. Nevertheless, beyond their data representation, IoT systems have the same data flow through their layers, so FDIA can be performed on all types of IoT systems. Based on this statement, the tool we introduced in this work should accept all types of data from all types of connected objects. It should also propose

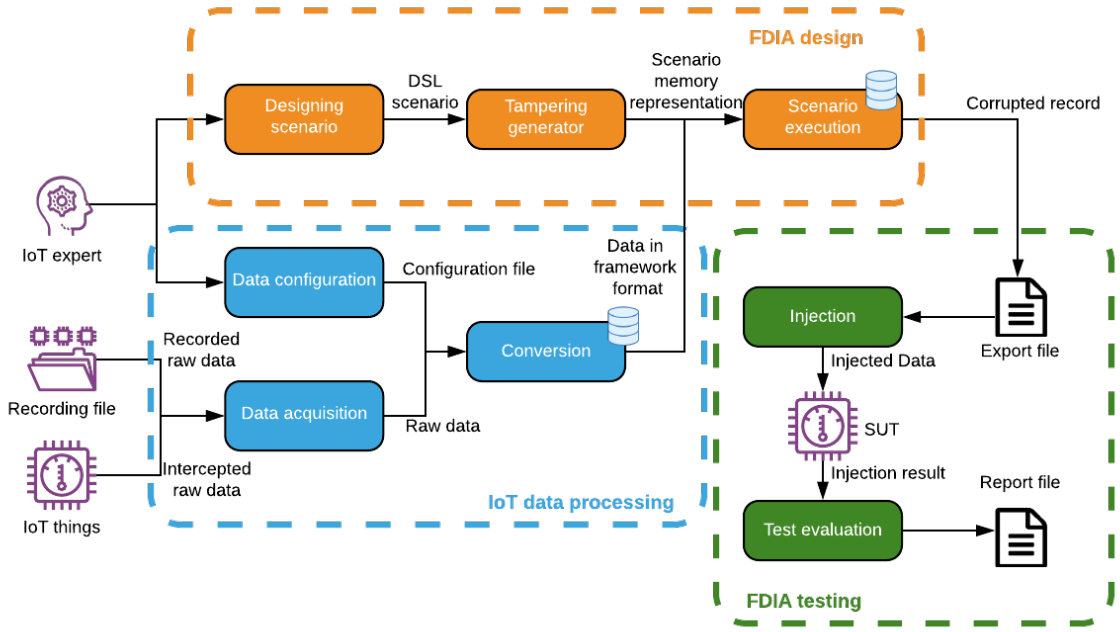


Figure 6.1: Workflow for FDIA testing - Data acquisition, designing and testing

data tampering tools for FDIA testing on all IoT systems, as well as tools for test injection and evaluation.

The workflow of our approach to address the FDIA challenge is shown on Fig. 6.1.

Data acquisition is the first step of this workflow, it consists of two things: either importing an IoT dataset under a file format such as a CSV or JSON, or intercept the IoT dataflow by performing a man in the middle or eavesdropping attack to perform a live attack.

Data configuration is made by the expert of the SUT. It consists in the property definition of the data present in the dataset, or in the flow defined in the data acquisition step. This is done using the information retrieved in the chapter 4.

The conversion is the step where input data are transformed into an internal format, for better data processing, and to handle a large heterogeneity of data and devices. These converted data are stored in a database.

Designing scenario is made by the expert of the SUT. We provide a textual DSL for designing a scenario of FDIA, and the expert uses it and selects the specific dataset he wants to alter.

Tampering generator is the phase where we read the scenario describing FDIA, and transform it into a memory representation.

Scenario execution is the process of applying the scenario written by the expert to the dataset. It results in a new dataset with tampered data stored in a database. After this step, the expert can choose to export this dataset, either for analysis, to inject into the

SUT, or to train machine learning tools to detect false data injected into the dataset.

Injection is the process of converting the data back to their original form and injecting them into the SUT.

The SUT represents the system where data altered by FDIA are injected, and where the test will be performed.

Test evaluation aims to verify the impact of the injection in the SUT with a test oracle. This test evaluation should be exported to a report for the purpose of either correcting the SUT, or improving the FDIA written by the test expert.

6.2/ DSL USAGE

In this section, we present the elements for gaining confidence in the proposed DSL. We will address the three main topics; the first is the data management, which is the basis of the FDIA attacks, the second is the validation of the implementation, and the last address the question of the expressive power of the language to describe the FDIA.

6.2.1/ DATA MANAGEMENT

As it has already been discussed above, data are the central element of the FDIA, and therefore of our approach. Over the years, there has been little clearly defined standardisation of IoT and, in any case, very limited use of it by manufacturers. This lack of standard is directly reflected in the data, whether in the format used, or in the structure itself. For the purpose of our approach, we would like to be able to carry out FDIA simulations on as many IoT as possible. Some requirements for the data management are therefore to be defined:

- Process data from as many IoT as possible: we take into account the nine kinds of sensors which can be embedded in IoT, presented in section 4.1.
- Enable data and format conversion between the most common IoT formats: we can support all formats identified in section 4.2, with the types (basic types: integer, float, boolean, string and also time), life cycle, and associated usage.
- Avoid losing information in the process: indeed, the input data of our tool must be able to be re-injected into the SUT without loss of information or format. Without this condition, it would be complicated to inject consistently the altered data into the SUT, both because of format compatibility, and because any intrusion detection tool would check the integrity and format of the input data and react accordingly.

This is why we have implemented a data management system that allows manipulation, regardless of the original format from the SUT. This allows data alteration to be made in our tool regardless of the format. The most efficient way to accept the majority of formats is to define a data format at the input of our tool. For this purpose, we decided to transform all input formats into a JSON format without structure, this is done using a flattening algorithm. The data structure is kept in the key part of the JSON key/value pair representation, which allows us to retrieve the original format from the tool output, thanks to a un-flattening algorithm.

This is illustrated in Fig. 6.2. The initial data format in Fig. 6.2a is flattened in the Fig. 6.2b. The hierarchical structure is preserved in the JSON key by separating the hierarchical levels with a separator symbol, here the ”dot” symbol. This makes it possible to reconstruct the structure later in the tool output, and to allow injection into the SUT. It should however be noted that for an implementation with a JSON format, the separator symbol should not be the ”dot” symbol, to avoid interfering with databases. In the following, the separator symbol will thus be the asterisk (*).

```
{"data": {
      "meter_code": "10",
      "temperatureTC": 8.03,
      "HumidityRH": 94.77,
      "LAeq":"6500",
      "noise":[0,2,23,26,...,44,33]
    }
}
```

(a) Initial JSON

```
{
 "data.meter_code": "10",
 "data.temperatureTC": 8.03,
 "data.HumidityRH": 94.77,
 "data.LAeq":"6500",
 "data.noise.0":0,
 "data.noise.1":2,
 "data.noise.2":23,
 "data.noise.3":26
 [...]
}
```

(b) JSON after flattening

Figure 6.2: Example of a JSON flattenning

Once the input file is in the right format, it has to be saved in a database in order to process the data efficiently. We chose to use a document-oriented database, which does not require a pre-established database structure, such as MongoDB[1]. These characteristics are interesting in our case, because we can process data of heterogeneous composition and structure. In this database, one mongoDB document represents one complete message of one IoT devices. Therefore, when querying particular message attributes, we can act on the whole message through its representative document. A structure is nevertheless present in each document in the database. The information contained in the messages is placed under a BSON[2] object named ”properties”. This allows the use of serialisation-deserialisation principles at the code level, which greatly facilitates the manipulation of documents and objects from the database.

[1] https://www.mongodb.com/
[2] Binary JSON: format for data storage and network transfer used by mongoDB

Since data management is now assured, we will turn our attention to the validation of the alterations.

6.2.2/ Validation of Alteration Primitive

To check the quality of our DSL in terms of its implementation and completeness of the test coverage, we need to verify that the actions requested by the DSL are well covered. To do this, we must test all the possible combinations that the DSL allows, and verify the results when interpreting them.

As we have seen above, a scenario action provided by the DSL consists of four main parts: the primitive, the selection criteria, the alteration criteria and the time frame, and each of these parts has several writing possibilities: the primitive part has 4 alternatives, the selection criteria part has 17 alternatives (if only one criterion is present), the alteration criteria part has 9 alternatives (if only one criterion is present), and the time frame has no alternative, as it is always present and always written in the same way. By combining all of these, this gives us 612 (4x17x9) different possibilities to define a single scenario action. Using a pair-wise method[3], we reduces it to 183 different possibilities to test, to cover 100% of the possibilities for defining a scenario action. These represent logical tests, which are implemented in physical tests by choosing concrete data to perform the tests.

For example, to test an action scenario with the primitive ”alter”, an equality with a string for the selection criteria, and a simple assignment of a decimal for alteration criteria, we should write the test scenario shown in Fig. 6.3.

The output of the test should produce a record where all messages with string identifier 10, and residing in the timestamp interval [0-4500], have the TemperatureTC property altered to 10.3.

```
1 scenario "failSensor"
2 ticker 2
3 alter things where identifier="10" set temperatureTC=10.3 from 0 to 4500;
```

Figure 6.3: Scenario example of a test

As another example, a scenario action with the primitive ”alter”, with for selection criteria, a selection inside a circle, and for alteration criteria, an evolution of a real value trough time, would results in the test scenario shown in Fig. 6.4.

The output of this test must produce a record where all messages coming from the devices located within a circle of 500m radius, and a centre defined by the latitude and

[3] http://www.pairwise.org/

longitude coordinates, have their particles property altered. This alteration starts at value 0 for the first message, and increases in steps of 10. If the number of messages selected is too large, this alteration is limited to value 99999.

```
scenario "ParticlesAlteration"
ticker 2
geolocation(47.213865,5.968195)
alter things where location isInsideCircle(47.213865,5.968195,500) set
    particles+=(0.0->99999.0,10.0) from 0 to 4500;
```

Figure 6.4: Scenario example of a test

The robustness concept is only focused on the expressivity of the grammar rules, and the compatibility of the objects in the database. The grammar will indeed reject any scenario that is not semantically recognised, which allows it to be robust to user input. Then, the robustness can intervene at the level of the types of data entered by the user in the scenario: the data selected or modified must be compatible with the data present in the base record file, and thus in the database.

If we take the same test example as in Fig. 6.3, we can partially modify it. This time, we do not focus on testing the correct execution of a scenario and its semantic acceptance, but rather a non-passing case, by altering a data with an inappropriate data type, for example, by altering the value of the property ”TemperatureTC” with a string rather than a number. This example is shown in Fig. 6.5. The output of this test should be the triggering of an error, as the scenario cannot be executed correctly.

```
scenario "failSensorTypeError"
ticker 2
alter things where identifier="10" set temperatureTC="10.3" from 0 to 4500;
```

Figure 6.5: Scenario example of a test

The tests to ensure the quality and the completeness of the DSL have been passed, we now need to ensure that the DSL is covered at a functional level, most notably in terms of the language itself and the data it processes.

6.2.3/ Coverage of the Knowledge

In this section we explore the coverage of the DSL scenario, particularly in terms of its expressivity, and therefore the levels of knowledge about FDIA that it is able to provide.

6.2.3.1/ Expressiveness and Domain Coverage of Primitive

As previously mentioned, IoT systems are very heterogeneous in their physical and technical aspects, but not only, as they are also completely heterogeneous at the functional level. For example, a FDIA targeting a smartgrid system will not have the same objectives, nor the same way of achieving them, as a FDIA targeting air traffic control devices.

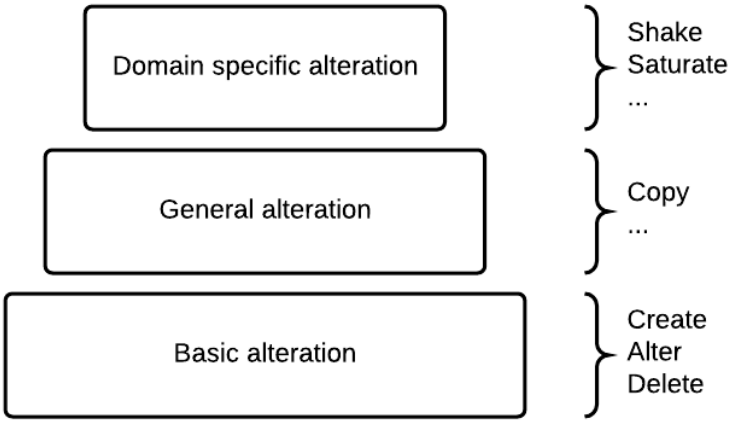


Figure 6.6: Alteration primitive level

For this reason, the basic expressivity coverage to be achieved is not related to the specific areas that can be found in IoT systems, but rather based on the basic functions of actions that can be performed on data: creating, modifying, and deleting them. We define for this purpose a three-level hierarchy of alterations according to their level of specialisation. This hierarchy is shown in Fig. 6.6.
In our language, and for all IoT devices, the basic data manipulation is well represented by the three alteration primitives: create, alter and delete. They are part of the basic alteration level in the hierarchy. The fourth alteration primitive, the copy, is a higher-level alteration primitive than the three basic ones, resulting from a specific alteration need related to the execution of a FDIA. However, as it is an alteration that can be performed on any type of IoT devices, it is present in the core language. It is represented as the level of the general alteration primitives in the hierarchy.
The third level in the hierarchy, the domain specific alteration, is dedicated to alterations linked to a specific domain. Indeed, the language accepts the addition of new alterations primitive, which can only be linked to a particular case study or field of use. To give an example of a hypothetical specific alteration, one could imagine a system of sensors around a volcano measuring seismic activity. An attacker might want to attack by simulating a seism, and our language would then be adaptable to handle a specific alteration called ”shake”, with specific properties such as the epicentre of the earthquake, or its magnitude. Of course, the specific execution of this primitive would need to also be coded. It

could hypothetically be represented in the form shown in the Fig. 6.7. This is a purely theoretical alteration, perhaps it would be more interesting, depending on the context, to create a custom specific alteration criterion, using the basic alteration primitive ”alter”.

```
scenario "earthquake"
ticker 2
shake things where location isInsideCircle(47.213865,5.968195,500) with
    epicentre(47.213865,5.968195) and magnitude=7 alter force from 0 to 4500;
```

Figure 6.7: Hypothetical scenario for a specific alteration

More concrete examples can be given on the basis of the scenarios identified in domains concerned by FDIA issues, such as ATC. In their work, [Cretin et al., 2020] discuss their method of generating FDIA-specific test data on the ADS-B protocol used in ATC, and have identified attack scenarios that can be applied to their areas of expertise. They have identified six classes of FDIA scenario, which are ghost aircraft injection, ghost aircraft flooding, virtual trajectory modification, false alarm attack, aircraft disappearance, and aircraft spoofing. Some of these classes could be covered by our DSL natively, without implementing a new alteration primitive or alteration criteria.

We take as an example the scenario of an aircraft’s disappearance, where an attacker seeks to make it disappear completely from control systems, with the aim of disrupting collision avoidance systems, disrupting ground stations or grounding the attacked aircraft for safety checks. In practice, this attack is modelled simply by deleting all messages sent by the attacked aircraft. This scenario could be modelled in our DSL with the following action:

```
delete things where ICAO=39AC47 from 0 to 5000;
```

ICAO is the identification property of the aircraft in the messages.

Another FDIA scenario for ATC that could be handled natively by our DSL would be a false alarm attack. This scenario is common in many domains, it consists of modifying the messages sent by changing the value of an alarm field. This scenario could be modelled in our DSL with the following action scenario:

```
alter things where ICAO=39AC47 set SQUAWK=7500 from 0 to 5000;
```

Squawk is a property in which many codes can be registered, notably alarm codes. Here the code 7500 stands for a hijacked aircraft.

Nevertheless, in the scenarios identified, there are some that cannot be easily and efficiently managed in a native way by the DSL, and therefore require the definition of new alteration primitives, new selection criteria, and new alteration criteria: for example, a ghost aircraft flooding scenario, where several simulated aircraft are created in order to saturate the airspace around the selected one.

With our actual DSL this attack, including the creation of three aircraft, would look like this:

```
1 scenario "saturateWithoutSpecificPrimitive"
2 copy things where ICAO=39AC47 set ICAO=25AC48 and altitude+=1000 and longitude
      +=0.03 from 0 to 9999;
3 copy things where ICAO=39AC47 set ICAO=26AC49 and altitude+=-500 and latitude
      +=0.015 from 0 to 9999;
4 copy things where ICAO=39AC47 set ICAO=27AC50 and altitude+=1600 and longitude
      +=0.03 and latitude+=0.026 from 0 to 9999;
```

With three aircraft, the objective is still attainable, but if the saturating attack needs much more of them, there is a need to define a new primitive specific for this attack scenario. This scenario also needs specific parameters, selection and alteration function for a correct realisation, which could take the following form:

```
1 scenario "saturateWithSpecificPrimitive"
2 saturate things where isInsideSphere(47.213865,5.968195,1000,500) set ICAO=
      random(icaoDB) with FloodAmount=3 from 0 to 9999;
```

The action is defined by the primitive named ”saturate”, followed by a new geographical selection (inside a sphere). It may be interesting to develop other types of geographical selections for this specific domain and then define an alteration criterion: here we introduce a notion of randomness for the identification of the simulated aircraft, followed by a parameter used for the action, which is the number of aircraft to simulate for each selected aircraft. As the scenario is already feasible with the native DSL, the new implementation can be easily added. Obviously, many functions have to be developed to manage the specificities of an aircraft, such as its altitude, speed, and direction.

This demonstrates that our DSL allows us to cover basic and general alterations that could occur in an IoT system attacked by FDIA. We can also extend it with more complex and domain-specific alterations, by defining new alteration primitives and new alteration functions targeted to that domain. This gives us good confidence in the expressiveness of our DSL, and in the following, we now explore the data coverage allowed by the DSL.

6.2.3.2/ Coverage of the Data

After exploring the coverage of language domains through alteration primitives, we need to look at the coverage of domains through data. This part provides additional information to the elements presented in the section 6.2.1 *”Data Management”*.

As defined above, data are at the heart of the approach, and are, by nature of the IoT devices, completely different between each different domains. Several elements thus have to be defined to allow the most complete possible coverage of the IoT domains that

may exist.

- Using a common data format: By deciding to use an input format specific to our tool, we allow compatibility with the majority of data formats that may exist in the IoT universe. All we have to do is convert the data between the SUT formats and the tool format.
- Using a lossless data format: The use of a lossless format is essential. This allows the tool's data, once modified, to be recovered in a format compatible with its original systems. For this purpose, the input format is a format without data structure (flattening), that retains the original structure in the property names, which a simple algorithm can implement.
- Using a data-independent storage system: By using a database that does not use a predefined data structure and is document-oriented, it is possible to store data message by message, as sent by their producing devices, without worrying about the size of the message, nor about the properties composing it. It allows an easier and more efficient processing of data than a relational database management system.
- Configuring the data: Some elements still need to be kept in a configuration for more efficient data processing. Configuring the type of each property that can be found within a message allows for more efficient processing during the alteration phase. Furthermore, the indication of a category to which the data belong to allow the tool to be prepared for any further processing, such as the use of machine learning methods.

All these elements allow us to accept data from the vast majority of IoT systems, regardless of their domain.

6.2.3.3/ Demonstration of the Interest

It is now worth exploring the value of the approach and the efficiency it can bring. We want to know how interesting a tool such as this one is to carry out FDIA tests, how easy it is to express a FDIA with this tool, to what extent it allows the user to write complex alterations in a simple way, and in short, how much more interesting it is to use a tool such as this one than to generate FDIA data by hand.

We now take a recording file from an IoT system with three devices sending a message every 15 minutes. These messages, converted to the internal format of the tool, have 101 properties. This recording, made during one month from July 1, 2019, to July 31, 2019, consists of 8931 messages.

The scenario created for the demonstration and applied to this recording is shown in the Fig. 6.8.

```
1 scenario "ApproachInterest"
2 ticker 2
3 geolocation(47.213865,5.968195)
4 alter things where location isInsideCircle(47.213865,5.968195,500) set
      particles+=(0.0->99999.0,10.0) with attenuation of 10.0 from 0 to 99999999;
5 alter things where identifier="10" set temperatureTC *= 1.1 from 0 to 99999999;
```

Figure 6.8: Hypothetical Scenario for a specific alteration

The scenario is composed of two scenario actions. The first action is an alteration where all the devices present within a defined circle, undergo an increase in the value of the particles detected by the sensors, this value is proportional to the distance from the scenario application point. The second action is an alteration that multiplies by 1.1 the value of the temperature measured by the sensor of the device with the ”10” identifier.

After execution this scenario, having only two actions, acts on the totality of the 8931 scenario messages, and alter 11908 different properties. The scenario took about 3.7 seconds to copy the original data, compute the altered data and then publish the results in a database, on a laptop with an i7-8665U CPU. The benefit of using this tool as well as this language, to generate FDIA attacks for testing purposes, takes here its full meaning, as it would be unthinkable for someone to perform this kind of tasks by hand; this would be even more obvious in more complex scenarios.

We will have the opportunity to see different scenarios, and to show the efficiency of the approach in more detail in the chapter on experiments.

6.3/ CONCLUSION

This chapter has allowed us to present the overall approach and the tool we have defined to meet the FDIA challenge, firstly, by presenting the workflow of the step-by-step approach of our work, and secondly, by providing elements to establish confidence in the expressive power and quality of the treatments proposed by the tool. To do this, we first studied the data management of our tool, and then saw how we were able to adapt to the heterogeneous formats and data of the IoT.

We ended this chapter on the coverage and power that the tool offers us. We first saw the coverage offered by the alteration primitives, and the fact that we can actually process and scale the DSL to cover specific domains. Then, we saw that we are able to accept data from several domains. Finally, we have shown the interest of using a method and a tool like this to perform FDIA, as soon as the amount of data increases, and also the

attacks become more complex.

In the next chapter, we will further investigate the effectiveness of the approach, and the expressive power of the tool, by using it in experiments.

III

Experimentation and Conclusion

7

EXPERIMENTATION

Contents

7.1 FDIA on Industrial Context 84
7.1.1 Simple FDIA - Sensor Failure 84
7.1.2 Mutli-Sensor FDIA - Attack on Particles Sensors 85
7.1.3 Private Companies 87
7.1.4 Municipality 88
7.1.5 Law Enforcement 88
7.1.6 Devices User 89
7.1.7 Other Services 89
7.1.8 Devices Manufacturers and Operators 89
7.2 Performance and Efficiency 90
7.3 FDIA Experimentation Synthesis 92
7.4 Machine Learning Experimentation 93
7.4.1 Data Synthesis 93
7.4.2 Detection of Attacks 95
7.4.3 Discussion 97
7.5 Conclusion 98

In this chapter, we present two main types of work we carried out. The first is the development of scenarios performed in the company context. We present different attack scenarios applied to the different parts of the company's systems, which allows us to show the expressive power and the possibilities of the scenarios. The second is the experimentation we conducted during the ANR project called GeLead. We present the joint works carried out with another team to generate attacked data, and to develop FDIA detection tools. This is achieved through the use of ML methods.

7.1/ FDIA ON INDUSTRIAL CONTEXT

To illustrate our work, we present eight attack scenarios. In order to show its diversity, we start with two generalist scenarios, aiming at a better understanding of the data handled and of the general approach, and then we use the six actors identified in the Fig. 2.1, creating for each one a FDIA scenario showing the range of possibilities.

In the different examples, we use a real IoT system, currently in operation in the streets of several cities around the world. This system is used to monitor environmental conditions within these cities, such as temperature, humidity, noise, particle concentration and nitrogen dioxide concentration.

The data scheme returned by the devices is given in Fig. 7.1. Data are retrieved every 15 minutes and sent immediately when the network is available.

```
{"data": {
      "temperatureTC": 8.03,
      "HumidityRH": 94.77,
      "LAeq": 6500,
      "No2": "24",
      "noise": [0,2,23,26,.....,44,33,22],
      "particles": 18939,
      "location": "47.213865,5.968195"
    }
 "meter_code": "10",
 "timestamp": 637458300
}
```

Figure 7.1: Data gathered by a thing in JSON

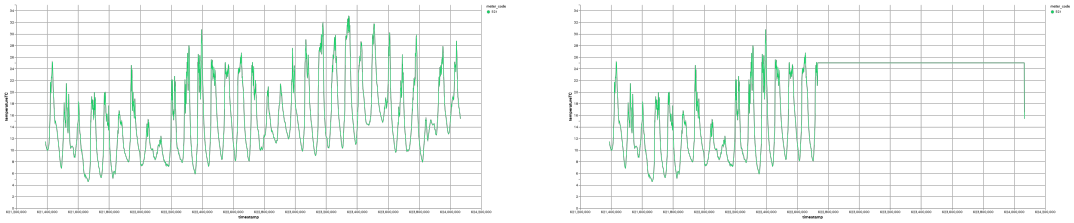
In the following experimentation, we work on data files extracted from the aforementioned system, these files represent one month's worth of records.

7.1.1/ SIMPLE FDIA - SENSOR FAILURE

In this experimentation, we perform a simple FDIA to simulate a sensor malfunction or failure. This type of attack creates a basic disruption which is easily detected by a human, leading to costly maintenance decisions for the system operator, in addition to weaken the trust of system operators and users.

Figure 7.2a shows the temperature data in Celsius in May 2019. Each spike in the line chart corresponds to one day. We apply the DSL scenario shown in Fig. 7.3 to this dataset. The scenario describes an alteration where all the data in the time window defined are replaced by the value of 50 degrees Celsius. Fig. 7.2b shows the execution result. Any human, as well as any algorithm, could easily detect this attack.

With this experimentation we validate the simple selection and alteration of the grammar.



(a) Initial temperature (b) Altered data

Figure 7.2: FDIA on temperature in degree Celsius in May 2019

We can produce the same kind of alteration on other data of the example, such as humidity to simulate the rain, noise to simulate noise pollution, or the number of particles and NO_2 concentration in the air, to simulate the absence of pollution, or a traffic jam.

```
1 scenario "failsensor"
2 ticker 2
3 alter things where meter_code="521" set data*temperatureTC=50 from 622732500 to
      624066300;
```

Figure 7.3: FDIA for sensor disruption

7.1.2/ Mutli-Sensor FDIA - Attack on Particles Sensors

In this experimentation, we perform a FDIA with a more complex selection and alteration function. The objective of this attack is to select multiple things using their geographical location, and simulate a gradual increase in pm10 (particles between 2.5 and 10 micrometres) particle levels in a part of the city. To simulate a more realistic attack, we need to select several sensors by their geolocation, and apply the attenuation of the attack magnitude according to their distance from the attack. This type of attack can aim at several objectives, for example triggering alternating car traffic during pollution peaks, or lowering real estate value in a neighbourhood.

For this experiment, we use data from the pm10 particles sensor gathered during the month of May 2019 by three things identified by their meter_code identifier: 500 (green), 515 (blue) and 521 (yellow). The charted data is shown in Fig. 7.5.

The scenario described in Fig. 7.6 selects the three things through their geographical location: we choose to use the GPS coordinate of the thing with identifier 500 with a radius distance of 500 metres, for the circle selection. With this selection criterion we thus pick all the things shown on the map shown in Fig. 7.4, and we apply a gradual increase to the selection, that begins at 1 and increases over time with a step of 10. An attenuation of 10 by meter is applied to this alteration, according to the distance from the application point of the scenario.

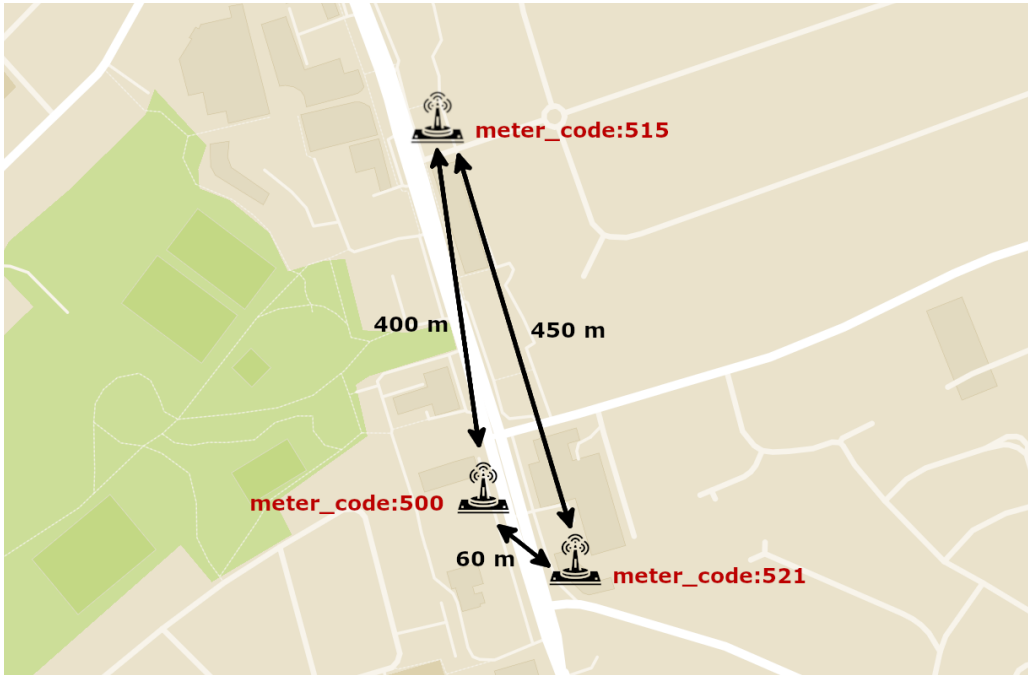


Figure 7.4: Map of the sensor situation

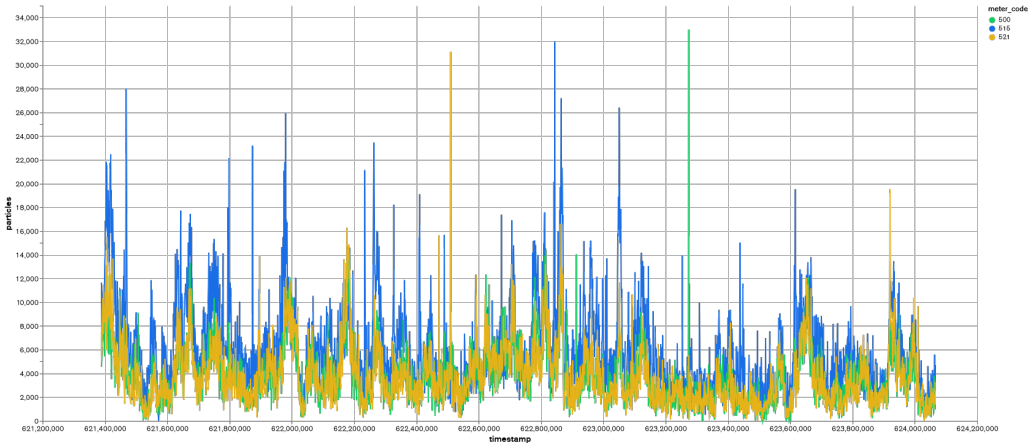


Figure 7.5: Particles in may 2019 before an FDI attack

Figure 7.7 shows the attack result: we can see the effects on the three curves through the progressive increase of their values, and we also see that the farther the object is from sensor 500 (center of the attack), the lower the value of its curve is. In the initial data, sensor 515 was predominant in terms of particle number, and sensors 500 and 521 were almost following the same trend. After the attack, the sensor closest to the attack (sensor 500) is predominant over the two others. Sensor 515, which is the furthest away, detects fewer particles, and this is visible on the figure, since its curve is below the others.

The expressiveness of the approach allows us to define this type of complex attack on several sensors. By modifying these scenarios, we can for example, simulate the propagation of sound in the city, or simulate the increase of heat and humidity levels in the city

```
scenario "IncrementationAndAttenuation"
ticker 2
geolocation (47.213865,5.968195)
alter things where data*location isInsideCircle(47.213865,5.968195,500) set
    data*particles+=(0.0->99999.0,10.0) with attenuation of 10.0 from 0 to
    99999999;
```

Figure 7.6: FDIA for particle increase with distance attenuation

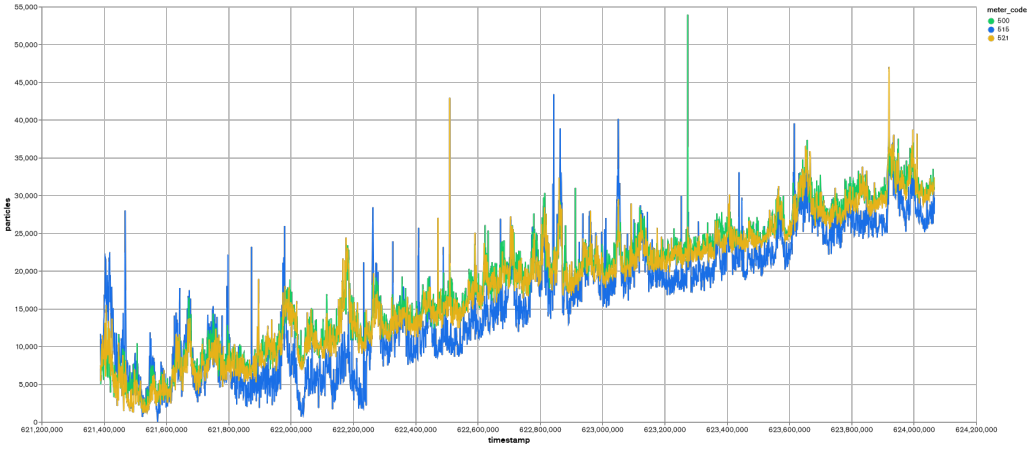


Figure 7.7: Particles in may 2019 after the FDIA execution

to trigger a heatwave plan.

To demonstrate the expressive capabilities of the DSL, we take the different parties involved in our systems, as shown at the beginning of this document in Fig. 2.1, and give an example of an attack scenario which might affect them. Banking elements do not fall within the scope of our experiments.

In the next six examples, we present an attack scenario related to the business in regards of the actors identified, in order to illustrate the facility with which it can be expressed.

7.1.3/ PRIVATE COMPANIES

The parking devices are not used solely to provide parking services; private companies use them to provide many other services, one of them being the distribution of gas cylinders.

Several attacks are likely to be carried out on this type of system, however in the case of a FDIA, the most probable one's concern data involving time: for example, sales statistics, which allow the deduction of the remaining stock. Such an attack could take the following form:

```
scenario "GasCylinder"
delete things where identifier=5 and messageType="sale" from 0 to 9999;
```

This scenario indicates the deletion of all messages related to the purpose of selling, from a particular selling device. Eventually, the stock will fall to zero, and the customer will buy empty gas cylinders, creating a loss of confidence in the system and the company. Artificial sales could be created the other way around, which would result in unnecessary stock reloading, and thus unnecessary costs for the company.

7.1.4/ MUNICIPALITY

The main use of the IPS by municipalities is for parking services. However, many other services can be used, and many attacks can therefore take place, depending on the service targeted: for example, environmental data services are very interesting to municipalities, in the context of smart cities. The attack example in section 7.1.2 illustrates an attack on particle data, that can lead to unwanted actions, and harmful effects such as alternating the traffic, or lowering traffic speeds.

Concerning the IPS main use, which is parking, attacks will mainly be focused on parking rights transactions, and alarms within the devices. Such an attack could take the following form:

```
1 scenario "ParkingRight"
2 alter things where identifier=5 and messageType="parkingRight" set endOfRight
    +=250 from 0 to 9999;
```

This scenario modifies the outgoing parking right creation messages from the device. The change affects the end of the parking right, by increasing the expected duration. The consequences are of economic nature for the municipality, mainly in terms of parking violations.

7.1.5/ LAW ENFORCEMENT

Law enforcement agencies are also users of the services produced by the terminals, mainly to enforce parking time limitations within cities. Nowadays the control is done through licence plates, it is compulsory in some countries, and in strong development throughout the world. For example, an alteration of the licence plates number on the parking meters, or in the smartphones of the control officers, could stop enforcement. Other services can also be used by law enforcement agencies, for example noise measurement services to detect abnormal noise in the streets of a city such as gunshots or shouting; by modifying the measured noise histogram, some noise can be added and some can be removed. Such an attack could take the following form:

```
1 scenario "NoiseAttack"
2 alter things where meter_code=5 set array(data*noise*,60,77)=0 from 0 to 9999;
```

This FDIA scenario leads to the suppression of all noises above 95 decibels, removing the possibility of alerts when loud noises are detected.

7.1.6/ Devices User

Devices users are the final users of the services. They can fall in many different categories, and are generally data producers and receivers. In the case of parking devices, it is the user who wishes to obtain parking rights for his vehicle; an attack can, for example, target the transaction, to make it disappear, and thus prevent a proper registration or ticket issuance. Such an attack could take the following form:

```
1 scenario "parkingRightRemover"
2 delete things where identifier=5 and messageType="parkingRight" from 0 to 9999;
```

This scenario deletes all messages from devices whose identifier is 5 and whose message type is a parking right request.

7.1.7/ Other Services

Obviously, many other services exist, and many of them use data that could be used to conduct a FDIA. For example, devices may switch to slave mode, in a master-slave dynamic with the data centre, to provide more processing-intensive or occasional and unusual services, such as weather-related services, local information services, or coupon services to obtain discounts; these services may also be affected by FDIA. The following scenario shows an example of attack on these type of services.

```
1 scenario "couponAttack"
2 alter things where identifier=5 and messageType="DiscountCode" set
    discountValue+=10 from 0 to 9999;
```

The coupon services are attacked by increasing the value of the original coupon by €10. This action is triggered by the message type identifier, and is applied to a single device identified by its number.

7.1.8/ Devices Manufacturers and Operators

The company that produces the devices may also be affected by FDIA, mainly in terms of reputation and economics.
Reputation: because any disruption to the services provided, necessarily leads to a deterioration in the manufacturer reputation.
Economics: because the services provided are mainly used for profit, the unavailability

and disruption of targeted services necessarily involve an economic loss. It can also take the form of unnecessary repetitive and costly maintenance.

All of the above attack scenarios can affect the company. The scenario presented in Section 7.1.1 shows a good example of a simple attack that results in economic loss, through unnecessary maintenance. In this case, the maintenance department will have to come and check the condition of the sensor, aiming to clean or replace it.

In the next section, we explore this approach through its efficiency.

7.2/ Performance and Efficiency

IoT systems have a huge amount of data flowing through their architectures: from a few messages per minute to several per second, the amount of messages that a FDIA attack must alter can quickly become problematic. Moreover, a message can contain several dozen measurements, so an attack can, depending on the system, alter large amounts of data. To demonstrate the performance of such an approach, we will measure the execution time of attack scenarios on several datasets.

We still use the same environmental sensor system, with a dataset of just over a year's worth of data, from 19 March 2019 to 31 March 2020. This dataset contains a total of 94812 messages, each message containing 88 interesting data to exploit in a FDIA.

All tests are performed on a developer laptop running Windows 10, with an Intel i7-8665U (1.90Ghz) CPU, 16.0GB of RAM, and an SSD. For greater accuracy each test is also run 10 times, and the reported value is the average of these runs. The execution time is measured as soon as the scenario starts, and stops at the end of the writes to the database.
In order not to lose the original data, the execution process makes a copy of the mongoDB collection that will be modified. This copy process was done on the same dataset for all our tests, and took an average of 3.2 seconds to run before moving on to the actual execution of the scenario.

The data and results of the effectiveness tests are given in Table 7.1. The altered messages give the number of messages impacted by alteration, the altered data give the total number of data modified in all altered messages, and the execution time is the elapsed time between the execution of scenario and the entry of the generated altered messages in the database.

In the following, we give the scenario use in each test.

Test 1: few messages and few data by messages

Alteration of the temperature data of the first 10930 messages of the dataset.

Test ID	Altered messages	Altered data	Execution time
1	10 930	10 930	3.7s
2	10 930	841 610	4.7s
3	94 812	94 812	4.8s
4	94 812	7 300 524	14.7s

Table 7.1: Overview of efficiency experiments

```
1 scenario "Efficiency1"
2 alter things where meter_code="515" set data*temperatureTC+=50.0 from 0 to
      621000000;
3 alter things where meter_code="500" set data*temperatureTC+=50.0 from 0 to
      621000000;
4 alter things where meter_code="521" set data*temperatureTC+=50.0 from 0 to
      621000000;
```

Test 2: few messages and many data by messages.

Alteration of the 77 noise data of the first 10930 messages of the dataset

```
1 scenario "Efficiency2"
2 alter things where meter_code="515" set array(data*noise*,0,77)+=50.0 from 0 to
       621000000;
3 alter things where meter_code="500" set array(data*noise*,0,77)+=50.0 from 0 to
       621000000;
4 alter things where meter_code="521" set array(data*noise*,0,77)+=50.0 from 0 to
       621000000;
```

Test 3: many messages and few data by messages.

Alteration of one data, the temperature data of all messages of the dataset.

```
1 scenario "Efficiency3"
2 alter things where meter_code="515" set data*temperatureTC+=50.0 from 0 to
      999999999;
3 alter things where meter_code="500" set data*temperatureTC+=50.0 from 0 to
      999999999;
4 alter things where meter_code="521" set data*temperatureTC+=50.0 from 0 to
      999999999;
```

Test 4: many messages and many data by messages.

Alteration of the 77 noise data of all messages of the dataset.

```
1 scenario "Efficiency4"
2 alter things where meter_code="515" set array(data*noise*,0,77)+=50.0 from 0 to
       999999999;
3 alter things where meter_code="500" set array(data*noise*,0,77)+=50.0 from 0 to
       999999999;
```

```
4 alter things where meter_code="521" set array(data*noise*,0,77)+=50.0 from 0 to
      999999999;
```

These few tests, on various amounts of altered messages and data, show the efficiency of the approach. Performing this kind of task without the help of a specialised tool is unthinkable. The time required to test this kind of attack would be far too high to make the testing sufficiently profitable. The approach benefits are not limited to technical performance: the rapid replay of scenarios is an interesting point for efficient testing, especially by subtle and rapid modifications of the altered properties in the attack model. This allows multiple datasets to be efficiently generated to functionally test similar system properties. It can be seen that the alteration time is linear with the number of data items altered, as the efficiency will be much more dependent on the complexity of the alteration. For example, the higher the number of actions indicated in the scenario, the longer the execution time will be.
Given the efficiency of the approach on an offline (indirect) SUT, it would be quite possible to imagine an efficient transition of the tool to an online (direct) approach, that is to say, altering the data on the fly at the input of the tool and reinjecting it into the SUT immediately. The tool would then be in the same conditions as an attacker performing a MITM attack.
The point that still needs to be explored further in terms of efficiency, is the time taken to define the attack scenario, that is whether or not the expert in charge of the modelling is able to model attacks efficiently enough to keep the approach interesting. This is answered by the fact that there is no equivalent approach to modelling and testing: modelling will therefore always be more interesting than performing the alterations by hand.

7.3/ FDIA Experimentation Synthesis

Through this section, we have been able to explore many different attack scenarios, which can target all actors in the data chain. Of course, many other scenarios can exist in a complex system such as IPS. Thanks to this approach, hundreds of attack scenarios targeting individually or together dozens of sensors, alarms, or services can be modelled to produce test data of FDIA in real life systems. Through our experiments, we noticed that most of the company's IoT-related systems, and the actors involved in these systems, were subject to FDIA risks. Our DSL allowed us to model and simulate attacks on these systems, and therefore met their testing needs. All types of data encountered are covered, and all the basic modelling needs are covered as well, which allows us to model and run FDIA on all IoT related systems. During these experiments, we also encountered scenarios requiring more advanced language functions. Therefore, future improvements

are expected to cover more than just the basic notions of FDIA.
Among these scenarios, two general types of scenarios emerge: attacks on time series data, and attacks on ”one-time” data. The first seek to make the system evolve to a certain state to satisfy the attack objective, whereas the second also modifies time-related data, but does not require the system to evolve. The benefits of the DSL its associated tool is mainly focused on the former, especially as it is more difficult to model and detect such FDIA.
We have also shown through several tests the efficiency of our approach, and thus the benefit it brings to the modelling and testing of FDIA.

In the next section, we experiment with ML, particularly in the domain of attack detection.

7.4/ Machine Learning Experimentation

In this section, we explore a joint work carried out with another research team, which study the ML aspect in the Gelead ANR project, to investigate the challenge of detecting FDIA with ML techniques.

7.4.1/ Data Synthesis

The work was carried out with a team from the DISC department of Belfort city. Based on the sensor data we provided them, they looked for ways to detect whether a FDIA might have occurred on a dataset. They decided to use machine learning methods, and in particular prediction methods.

In order to do this, they first looked at the nature of the data available, seeing how consistent the data were, and how they could be analysed for predictive purposes. Indeed, to be used in prediction cases, the data must not be chaotic, and need patterns and correlation in order to be able to infer information from it.

Out of the six measurements made by the sensors (temperature, humidity, global sound levels, nitrogen dioxide, noise and particles level), one type of data has proved to be more interesting than the others to predict and then detect, these are the noise data. As a reminder, the noise data are continuously collected by a sensor, and sent every 15 minutes. They are presented in the form of an array of 77 values, where each value is associated to a class; each class represents a decibel threshold reached during the 15 minutes: the first class is the threshold of 35 decibels, and the last class is 112 decibels, with a step of 1 decibel by class. For example, the fifth value in the array represents the number of times the noise reached 40 decibels, during the 15 minutes of measurement.

Figure 7.8 shows the noise record over a period of approximately one month, where

we also have calculated the average noise measured per message. We add for this purpose the 77 value classes together, multiplied by their corresponding decibel values, and divided by the total number of measurements made: this gives us the average noise recorded during the 15-minute period in decibels. We can see a visual pattern of the alternation of day and night with the rise and fall of the average noise value; we can also distinguish weekends, which are less noisy than weekdays.

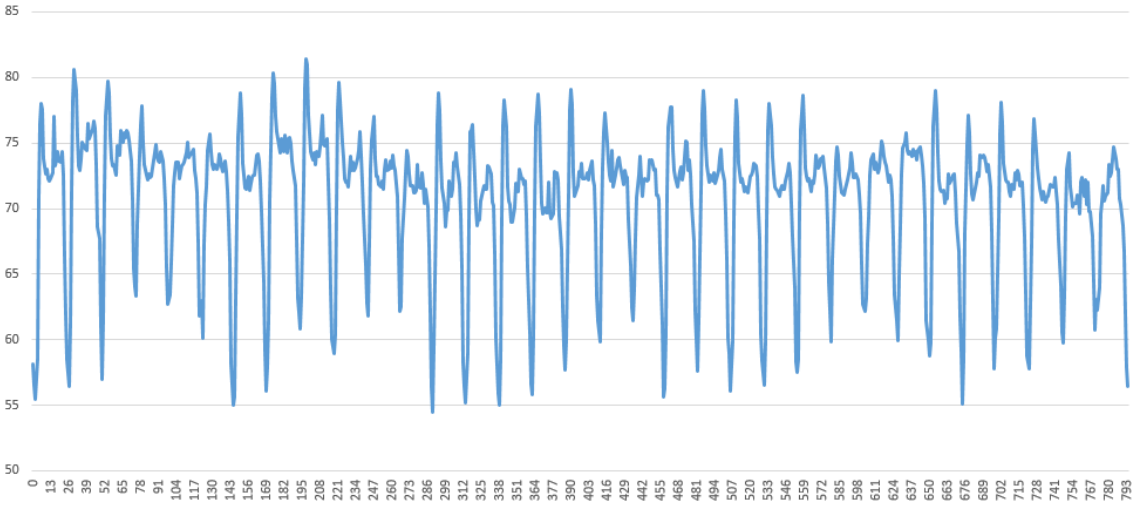


Figure 7.8: Average noise in decibels

These visual patterns show that data prediction is possible. For this purpose, the Belfort team has developed a script to perform machine learning on these data using two learning methods, the first one with neural network CNN-LSTM, and the second one with gradient boosting LightGBM. In Fig.7.9 is shown the training with the lightGBM method. The blue curve represents the actual training data, the orange curve represents the training itself and the green curve represents the prediction made on data not encountered during training. In Fig. 7.10 we can see the prediction part which follows well the unknown data curve.

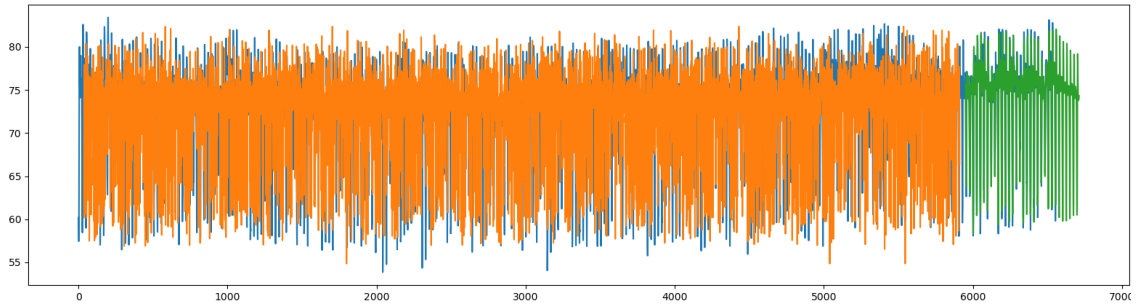


Figure 7.9: lightGBM training

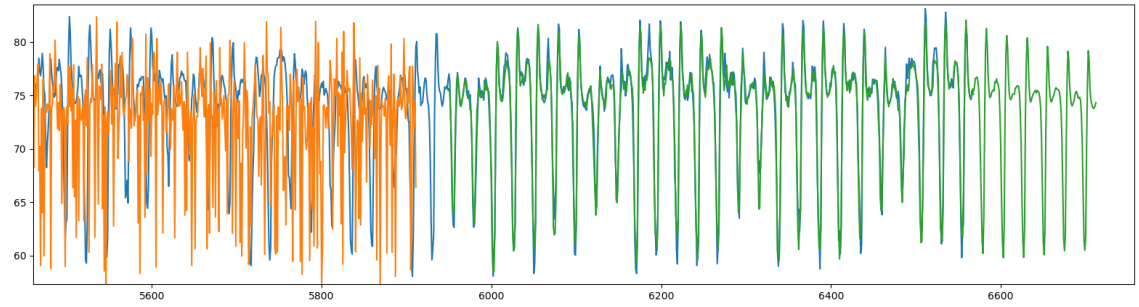


Figure 7.10: lightGBM training zoomed

7.4.2/ Detection of Attacks

To detect if an attack is occurring on a dataset, the prediction values are compared to the corresponding test data value; if the difference between the two values is too high, then this data is flagged as potentially attacked. This detection is shown in the Fig. 7.11. Here, data are changed randomly by increasing or decreasing their values, and we can see that 27 data are considered to be attacked. The attacked data are represented by the peaks on the green curve, and also visually by the unusual peaks on the blue curve. The peaks on the red curve represent an anomaly detection, i.e., a difference between the prediction and the controlled value that is too high. We can see that out of the 27 attacked data, 23 are detected, and 4 are not. It can also be noted that there are no false positives found during detection. Undetected attacked data are due to attacks that were too weak.

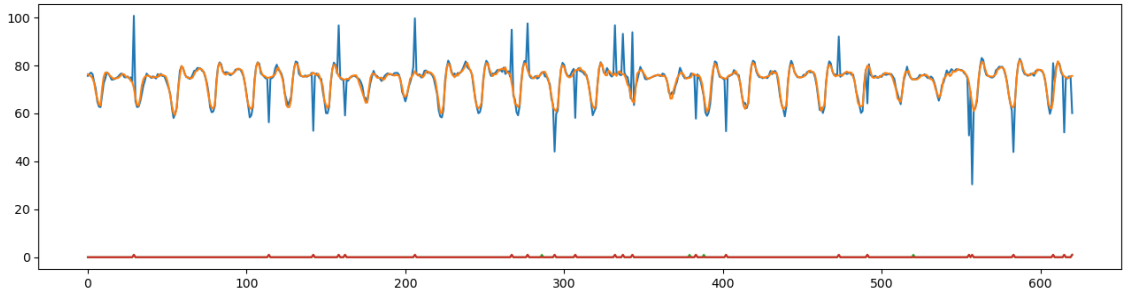


Figure 7.11: Detection with lightGBM on a random value attack

Obviously, this detection method is very effective in detecting anomalies: any slight variation between predicted and to-be-checked data is detected. We will now perform some FDIA attacks using our approach, to go further than a simple random value attack.

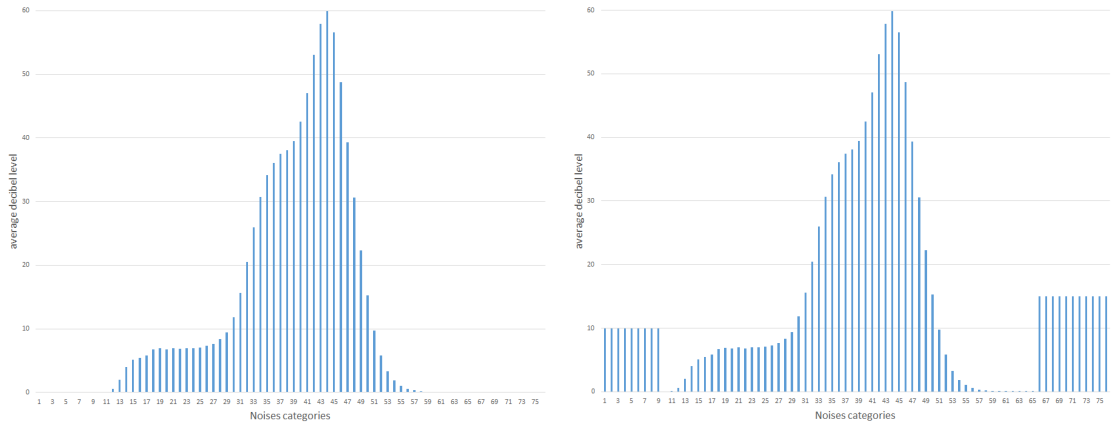
As mentioned above, the noise data are represented in a histogram of 77 values. The detection uses the average of these values; by manipulating the source data, the distribution of the measurements can be changed, without changing the mean. This makes it possible to run a FDIA, without being detected by this detection method. In the scenario in Fig. 7.12, we alter in the noise histogram the first ten values by performing an

increase of 10, and the last ten values by an increase of 15, covering the entire duration of the recording. This scenario aims to simulate very loud unusual noise (increase of +15 measurements on noise classes above 101 decibels), while cancelling them by simulating very weak noises (increase of +10 measurements on the noise classes higher than 35 decibels and lower than 45 decibels), in order not to modify the average noise of the recordings. All 657 recorded messages are therefore altered by this attack.

```
1 scenario "start10andend15"
2 alter things where meter_code=515 set array(data*noise*,0,10)+=10.0 and array(
      data*noise*,66,77)+=15.0 from 0 to 999999999;
```

Figure 7.12: Scenario for stealth noise attack

The average decibel distribution by noise category on authentic data is shown in Fig. 7.13a. The distribution after the execution of the scenario 7.12 is shown in Fig. 7.13b



(a) Before FDIA (b) After FDIA

Figure 7.13: Distribution of average decibels by noise categories before and after FDIA

Any humans or algorithms could detect the FDIA efficiently, as it is rather noticeable on the figures, but the ML detection method presented above is not able to detect this attack successfully. In Fig. 7.14, we can observe the results of the detection of this scenario: in blue the data to be verified, in orange the prediction, in green the data being altered, and in red the data being detected as altered. We notice that the totality of the data are altered, and that only two data points are considered to be attacked by the detection method.

The attack presented here is not subtle at all, and would be easily detectable by adding other detection systems, for example checking that the number of measurements is constant between each recorded message. Nevertheless, the attack could be much more subtle in its alteration of the data, while remaining within the detection tool expected averages. We can see that ML detection tools thus require the help of synthetic data to improve, and reach better levels of effectiveness.

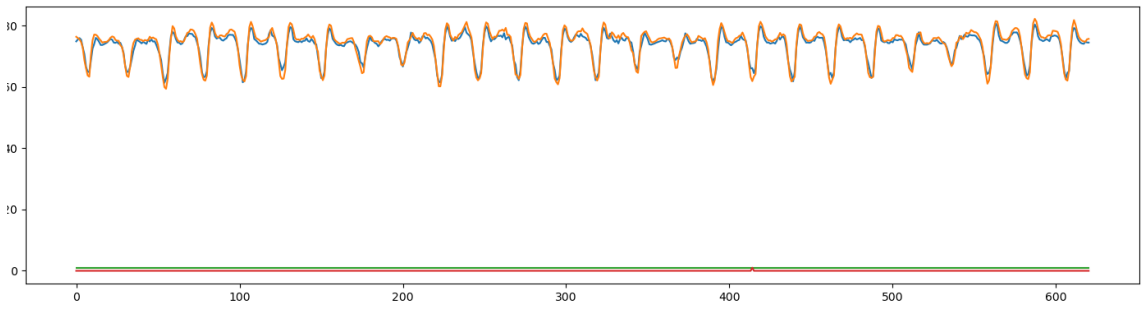


Figure 7.14: Detection of the attack on lowest and highest value of the noise histogram

7.4.3/ Discussion

We have seen that on specific non-chaotic data, this prediction is very efficient, and allows high detection of anomalies and attacks. As far as detection is concerned, we have seen that the efficiency, although perfect on noise data, is no longer so when we decide to attack the underlying raw data: detection must therefore be improved. It would be interesting to perform the training on several devices, for an anomaly detection using data from geographically close sensors. Also, as we have seen previously, one of the difficulties of working on FDIA is the small quantity of data being definitely considered as altered by a FDIA, particularly because they are difficult to detect. For the training phase of this detection method, it is therefore needed to have clean data, having never suffered any attack. Furthermore, an attack could aim specifically at altering the training data, and thus grant the attacker a greater freedom of attack choice, potentially undetected in the detection phase.

We also noticed that the recordings, although they show an ease of prediction, are not consistent over time. During March 2020, the first confinements were put in place to fight the COVID-19 pandemic, and we noticed the implementation of these confinements on our sensors, especially the noise sensors: the average noise recorded was reduced between the beginning and the end of March 2020. With the presented detection method, unattacked data like COVID-19 one could be detected as altered. This is shown in Fig. 7.15.

In order to have effective detection over time, it would be necessary to continuously redo the learning process with recent data. This would allow for gradual adaptation, as changes in the data distribution emerge. Nevertheless, this would allow an attacker to very subtly attack the data, and stay within the margin of error of the detection system, gradually bringing it closer to its attack objectives. The attacker could then carry out the desired attack without being detected, and could also slowly bring the data back to normal after the FDIA, so that the detection system would resume normal operation after the attack.

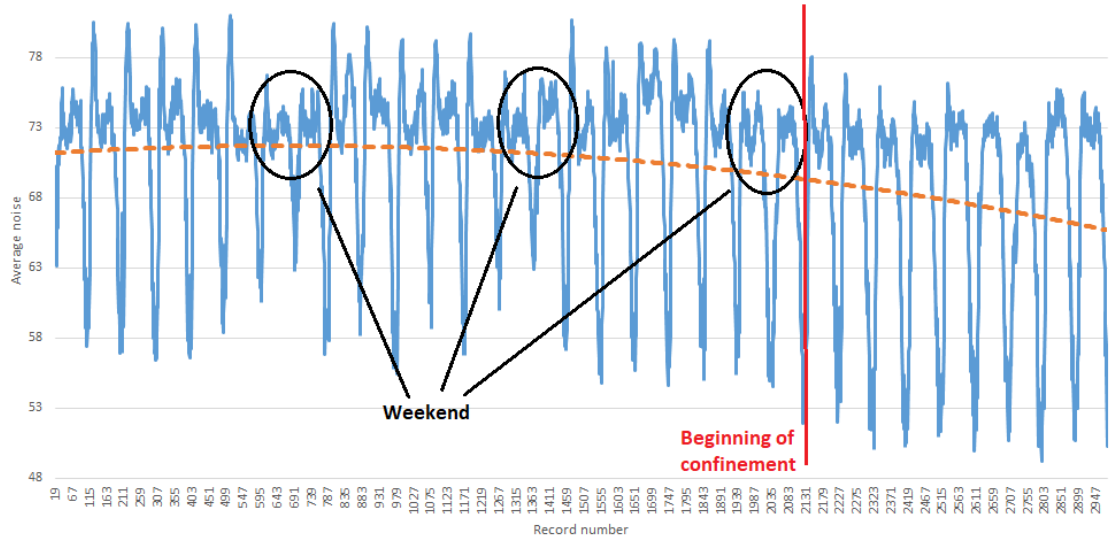


Figure 7.15: Average noise during the month of March 2020

The approach to FDIA detection using ML methods is still promising, and requires further research. Many works already use this kind of method to detect anomalies such as [Cordy et al., 2019; Mokhtari et al., 2021; Esmalifalak et al., 2017], and notably in areas sensitive to FDIA, such as SCADA or smart grid.

7.5/ CONCLUSION

In this chapter we have explored the approach by conducting experiments.

Firstly, we have described attacks on the environmental sensor systems currently used in parking meters, and then by describing attack scenarios that can affect the different stakeholders of the IPS systems.
Secondly, we explored the efficiency of the approach and of the tool, by measuring the execution speed in relation to the amount of data altered. The results show that it would be unthinkable to carry out this kind of attack by hand without using an approach like ours, and that our approach is therefore effective in solving the FDIA challenge.
Finally, we explored joint work with another laboratory on FDIA detection systems. Using machine learning methods, the noise data could be predicted with good effectiveness; this would allow for the detection of any anomalies that deviate slightly from the predicted data. We have shown that, although effective at first sight, this kind of detection system can be fooled by changing some underlying data. The natural modification of the distribution of the data could also lead to not being able to detect any anomaly efficiently. We also raised the problem of training on data that is free of any attack. To overcome the potential deviation of the distribution, training must be carried out regularly, but it can then

lead to attacks directly on the training data.

This chapter has allowed us to answer research question RQ3. At the efficiency level, there is no equivalent approach to solving the FDIA challenge; simulating such attacks directly by hand, even the simple ones, is far too cumbersome to perform. We can thus be confident in the usefulness of such an approach, and the speed of execution of the attacks, even on large datasets, allows us to validate the efficiency of the approach. In terms of functional issues, we were able to show that attacks on the diversity of stakeholder systems could be modelled and therefore addressed with our approach, giving us confidence in its effectiveness.

In the next chapter, we conclude this thesis with a summary and an outlook on future works that could improve our approach.

8

CONCLUSION

Contents

8.1 Summary 101
8.2 Future Works 103
8.2.1 Overall DSL Improvement 103
8.2.2 Testing Phase 104
8.2.3 Machine Learning Usage 105

This concluding chapter firstly presents a summary of the work presented in this thesis, and secondly explores the possible future works and research perspectives through three avenues of development.

8.1/ SUMMARY

In this thesis, the work took place in an industrial company called Flowbird, the world leader in parking solutions. We first analysed the systems present within the company and noticed that their architecture and general functioning were similar to those of the Internet of Things (IoT). Thus, these company systems are subject to the same security risks as those of the IoT.

Today, the ”things” of the IoT are omnipresent around us and use data from their environment to provide many services, which are more or less critical depending on their use. Their strong development over the last decade in both private and industrial spheres has led to the emergence of numerous security issues, particularly because of the lack of standardisation, and the large number of manufacturers using their own development methods that are not focused on security. One of these security issues is the False Data Injection Attack (FDIA). FDIA consist of the malicious modification of data transiting within an IoT system, especially at the level of the data production and manipulation entities. These modifications are carried out in subtle ways to deceive detection systems, whether

human or computerised. These subtle alterations are intended to bring the attacked system into a state where the behaviour of the system is modified. Obviously, unwanted changes in system behaviour is often critical. In the case of industrial systems, the risk range of such an attack is very large, ranging from life threatening, for health services, to reputational and economic loss for monetary services.
During the state-of-the-art phase, it appeared that the state of research on FDIA is well explored, especially for smartgrids, but that the state of research for IoT systems is poor. In addition, the majority of the work carried out focuses on the detection and filtration of data corrupted by FDIA. There are few works focusing on testing injection attacks, and none for IoT.

This work has therefore addressed the challenge of FDIA testing in the world of IoT. To this end, we have defined research questions that have guided our work.
The first one (RQ1) focused on the fundamental elements that constitute both IoT systems and FDIA: the sensors and their data. Indeed, the data are the elements coming from the sensors, that enable the various IoT services to be offered to users, and FDIA specifically targets this in order to modify their behaviour. We therefore need to study these data to understand their behaviour and use them in the attack modelling. To do so, we analysed the various sensors existing within and outside the company's systems to provide a classification. We then analysed the data that these sensors produce to extract their main characteristics. We thus identified two categorisations: one for the sensors, and one for characterising the data.
The second question (RQ2) concerned the process of modelling attacks. To answer this question, we have defined a Domain Specific Language (DSL) that allows us to model the attacks. We followed a multi-step methodology to explain why we chose to make a DSL, and to explain how it was designed. This DSL is part of a global approach in the three main steps which are data recovery and characterisation, attack modelling and execution, and finally testing the attack by injecting it. The tool implementing the approach allows currently the retrieval of the data from a majority of IoT systems, by using an internal format without loss of structure, to characterise the data, design attack scenario, execute the attack, and export the attacked data.
The third research question (RQ3) focused on the effectiveness of the approach defined to address the FDIA challenge. To answer this question, we tested the approach through experimentation. To do this, we used real systems, similar to IoT systems present in the company. We carried out several attack scenarios to demonstrate the interest of such an approach on such systems, in particular, by focusing on the expressiveness of the language and its effectiveness. Experiments with a detection system were also carried out, allowing us to highlight the benefits of our approach in the development of detection systems, as well as to evaluate the relevance, advantages and disadvantages of using such systems.

In conclusion, our approach is interesting for addressing the problems posed by FDIA in IoT systems. The use of a tool implementing our approach allows us to address the problem on two levels. The first one is to generate attacked test data, allowing to evaluate the robustness of systems to FDIA; the second one is to use the attacked data to train and asses detection systems. In terms of effectiveness, we can handle the majority of IoT systems regardless of their format. Firstly, thanks to the use of lossless formats internally in the tool, and secondly, thanks to a language that models at the IoT level and not the underlying domain, and has sufficient expressiveness to allow genericity of modelling. In terms of efficiency, the implementation of the approach is powerful and allows the execution of attacks quickly, thus enabling the processing of large systems with a lot of data. This makes the testing process efficient and economical.

To address the many different IoT systems in the company, we have focused the approach on its genericity. However, in the case of a very specific and complex IoT subdomain, our high-level approach may be less effective than an ad hoc tool: this is the main drawback. Nevertheless, the approach we have defined would be the same in a specialised tool, and the development effort of such a tool from scratch would be tedious. Moreover, our approach is designed to evolve and to be able to consider subdomains by enriching the language.

During this work , we kept in mind the evolution of the approach. We have therefore laid the foundations for future works to improve the approach, in order to address FDIA testing issues. The next section is dedicated to future works and the possible evolution of our work.

8.2/ FUTURE WORKS

The global approach we have defined and implemented during this thesis is functional and helps to overcome the FDIA challenge. Nevertheless, several improvements of our approach remain to be developed or would be interesting to develop. Therefore, in the next sections, we propose several ways of improving the approach.

8.2.1/ OVERALL DSL IMPROVEMENT

The DSL currently allows attacks to be modelled according to four basic primitives. Although these are sufficient to perform attacks on most systems, some attacks would be long and complicated to describe with these atomic primitives. Thus, other more complex primitives need to be developed to deal with more complex attacks and domains.

In terms of selection criteria, it would be interesting to set up a sublanguage dedicated to the evaluation of Boolean expressions. Currently, the DSL does not allow disjunction, which can be limiting for the modelling of certain attacks. Adding an evaluation language would fill this gap, while allowing the creation of other selection functions more easily and efficiently. A sub-language for arithmetic operations should also be implemented, allowing for greater expressiveness when modelling, especially when using variables.

At the level of scenario properties, many properties can be added depending on the operating domain of the IoT system. Nevertheless, integrating a system of variables would allow a greater ease of scenario modelling. It would also be interesting to use sets as variables: this would allow us to generate several attack models efficiently by performing the Cartesian product of these sets, reducing the overall modelling cost and increasing the test efficiency.

Adding the description of the expected outcome of the attack to the DSL would allow the approach to be used more comprehensively, up to the injection and test execution phase. This is further discussed in the next section.

8.2.2/ TESTING PHASE

Unfortunately, the third phase of the approach as shown in Fig. 6.1, the testing phase which consists of the injection of attacked test data in the systems under test, and the evaluation of the test, was not implemented in the tool to ensure automation. Currently, the test engineer has to export the results of the execution of the attack model. From this exported file of attacked data, he can perform various actions, including injecting the attacked data into the system under test to perform the various security tests that led to the modelling of the attack.
For a global implementation of the approach, the tool should allow tests to be run directly from itself. In a first instance, in an offline way: the data are altered at once, then injected it into the system under test, according to several injection parameters (time between messages, message orders, etc.). Then in a second instance, in an online way: the data are altered on the fly, that is when a message is intercepted and enters the tool, it is altered immediately according to the model, and then injected into the system under test. In order to carry out the tests efficiently and to inject the data directly from the tool, the integration of a connection with the System Under Test (SUT) is therefore essential: firstly a technical connection, to be adapted to the different protocols, technologies and methods used by the SUT, and secondly, a functional connection, to obtain information on the system during the test phase, in particular to know the effects of the tests and their results.
The technical connection would require studying the different protocols and technologies

used by the IoT systems, then implementing them within the tool. The functional connection would require studying the application domain, to extract the elements that will allow the tests to be carried out and to know their states effectively. However, due to the great heterogeneity of the systems this task would probably have to be done specifically for each system under test. The control of these connections could be described directly within the attack scenario, using the DSL, in particular through the use of Keyword Driven Testing (KDT). In [Bernard et al., 2020], the authors define a method using MBT techniques with KDT. This allows, thanks to an adaptation layer, the use of keywords to control the tests execution by describing the sequences of actions that will be performed on the system under test. At the end, the test scripts are then generated.

8.2.3/ Machine Learning Usage

The creation of attack scenarios relies entirely on the level of expertise that the attack designer possesses. A FDIA relies primarily on the functional characteristics of the system under attack, in particular the processes that dictate the behaviour of the system. The modelling of FDIA requires a deep knowledge of this type of attack, but also of the systems to be tested. The effectiveness of the attack and the tests are conditioned by the level of knowledge and competence of the test engineer. As we have seen, the use of Machine Learning (ML) methods is proving to be interesting to detect this type of attack, but not only: it would be interesting to use artificial intelligence algorithms to generate attacks and therefore effective tests. This would make it possible to avoid needing, in some cases, expert knowledge of the field being tested. The amount of relevant attack scenarios that could be generated would also be much higher, leading to better test coverage. In addition, a ML approach allows greater modelling efficiency than with a human modelling a few scenarios by hand.

In terms of detection, we have been able to explore a method that allows the prediction of non-chaotic values and therefore reveals attacks, as they are outside the prediction. During our experiments, we noticed correlations between the different data of the same message. For example, the sound data was correlated with the weather data: the predictions were better when using additional data. It would be interesting to develop detection methods using external data for better detection, the sensors in IoT networks are also often close to each other, and using this fine mesh would allow us to detect local anomalies more efficiently.

Finally, generation and detection do not necessarily have to be separated. It would be interesting to use them in common, to make them progress together. The use of Generative Adversarial Networks (GAN) would then be an interesting research topic in the case of FDIA. Introduced by [Goodfellow et al., 2014], GAN are ML frameworks in which

two neural networks compete against each other. One network generates artificial data from a training set, and the other network tries to detect whether the data are artificial or real. When a network succeeds in deceiving or detecting the other, the other can then progress by adding this data to its training set, leading to a mutual progression. A lot of research has already been done on the use of GAN in the security field, especially to improve intrusion detection systems [Dutta et al., 2020]. Moreover, as they are already used in the world of CPS for the detection of anomalies in time series [Li et al., 2019], we believe that the development of GAN to improve the generation and detection of FDIA could prove interesting.

BIBLIOGRAPHY

[Aazam et al. 2014] AAZAM, Mohammad ; KHAN, Imran ; ALSAFFAR, Aymen A. ; HUH, Eui-Nam: **"Cloud of Things: Integrating Internet of Things and Cloud Computing and the Issues Involved"**. In *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014*. Islamabad, Pakistan : IEEE, January 2014, pages 414–419. – ISBN 978-1-4799-2319-9. DOI: 10.1109/IBCAST.2014.6778179

[Albright et al. 2010] ALBRIGHT, David ; BRANNAN, Paul ; WALROND, Christina: *Did Stuxnet Take Out 1,000 Centrifuges at the Natanz Enrichment Plant?* 2010

[Analytics 2019] ANALYTICS, Strategy: *Internet of Things Now Numbers 22 Billion Devices But Where Is The Revenue?* 2019

[Anderson et al. 2014] ANDERSON, Jason W. ; KENNEDY, K. E. ; NGO, Linh B. ; LUCKOW, Andre ; APON, Amy W.: **"Synthetic data generation for the internet of things"**. In *2014 IEEE International Conference on Big Data (Big Data)*, IEEE, 2014, pages 171–176. – URL http://ieeexplore.ieee.org/document/7004228/. – Access date:: 2020-08-27. – ISBN 978-1-4799-5666-1. DOI: 10.1109/BigData.2014.7004228

[Aswale et al. 2019] ASWALE, Pramod ; SHUKLA, Aditi ; BHARATI, Pritam ; BHARAMBE, Shubham ; PALVE, Shekhar: **"An Overview of Internet of Things: Architecture, Protocols and Challenges"**. In SATAPATHY, Suresh C. (editors) ; JOSHI, Amit (editors): *Information and Communication Technology for Intelligent Systems* Volume 106. Springer Singapore, 2019, pages 299–308. – URL http://link.springer.com/10.1007/978-981-13-1742-2_29. – Access date:: 2020-07-27. – Series Title: Smart Innovation, Systems and Technologies. – ISBN 9789811317415 9789811317422. DOI: 10.1007/978-981-13-1742-2_29

[Barse et al. 2003] BARSE, E.L. ; KVARNSTROM, H. ; JOHNSON, E.: **"Synthesizing test data for fraud detection systems"**. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, IEEE, 2003, pages 384–394. – URL http://ieeexplore.ieee.org/document/1254343/. – Access date:: 2020-08-27. – ISBN 978-0-7695-2041-4. DOI: 10.1109/CSAC.2003.1254343

[Belkeziz and Jarir 2016] BELKEZIZ, Radia ; JARIR, Zahi: **"A survey on Internet of Things coordination"**. In *2016 Third International Conference on Systems of*

Collaboration (SysCo), IEEE, 2016, pages 1–6. – URL http://ieeexplore.ieee.org/document/7831328/. – Access date:: 2020-07-27. – ISBN 978-1-5090-4926-4. DOI: 10.1109/SYSCO.2016.7831328

[Bernard et al. 2020] BERNARD, Elodie ; AMBERT, Fabrice ; LEGEARD, Bruno: **"Supporting efficient test automation using lightweight MBT"**. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. Porto, Portugal : IEEE, October 2020, pages 84–94. – URL https://ieeexplore.ieee.org/document/9155866/. – Access date:: 2021-03-26. – ISBN 978-1-72811-075-2. DOI: 10.1109/ICSTW50294.2020.00028

[Bostami et al. 2019] BOSTAMI, Biozid ; AHMED, Mohiuddin ; CHOUDHURY, Salimur: **"False Data Injection Attacks in Internet of Things"**. In AL-TURJMAN, Fadi (editors): *Performability in Internet of Things*. Cham : Springer International Publishing, 2019, pages 47–58. – URL http://link.springer.com/10.1007/978-3-319-93557-7_4. – Access date:: 2020-05-06. – Series Title: EAI/Springer Innovations in Communication and Computing. – ISBN 978-3-319-93556-0 978-3-319-93557-7. DOI: 10.1007/978-3-319-93557-7_4

[Bray 2017] BRAY, Tim: *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. December 2017. – URL https://rfc-editor.org/rfc/rfc8259.txt

[Burhan et al. 2018] BURHAN, Muhammad ; REHMAN, Rana ; KHAN, Bilal ; KIM, Byung-Seo: **"IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey"**. 18 (2018), number 9, pages 2796. – URL http://www.mdpi.com/1424-8220/18/9/2796. – Access date:: 2020-01-06. – ISSN 1424-8220. DOI: 10.3390/s18092796

[Cai and Chen 2012] CAI, Liang ; CHEN, Hao: **"On the Practicality of Motion Based Keystroke Inference Attack"**. In *Trust and Trustworthy Computing* Volume 7344. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, pages 273–290. – ISBN 978-3-642-30920-5 978-3-642-30921-2. DOI: 10.1007/978-3-642-30921-2_16

[Chaojun et al. 2015] CHAOJUN, Gu ; JIRUTITIJAROEN, Panida ; MOTANI, Mehul: **"Detecting False Data Injection Attacks in AC State Estimation"**. 6 (2015), number 5, pages 2476–2483. – URL http://ieeexplore.ieee.org/document/7035067/. – Access date:: 2020-08-10. – ISSN 1949-3053, 1949-3061. DOI: 10.1109/TSG.2015.2388545

[Chomsky 1956] CHOMSKY, N.: **"Three models for the description of language"**. In *IRE Transactions on Information Theory* 2 (1956), number 3, pages 113–124

[Choudhary and Jain 2016] CHOUDHARY, Gaurav ; JAIN, A.K.: **"Internet of Things: A survey on architecture, technologies, protocols and challenges"**. In *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, IEEE,

2016, pages 1–8. – URL http://ieeexplore.ieee.org/document/7939537/. – Access date:: 2020-07-23. – ISBN 978-1-5090-2807-8. DOI: 10.1109/ICRAIE.2016.7939537

[Cordy et al. 2019] CORDY, Maxime ; MULLER, Steve ; PAPADAKIS, Mike ; LE TRAON, Yves: **"Search-Based Test and Improvement of Machine-Learning-Based Anomaly Detection Systems"**. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA : Association for Computing Machinery, 2019 (ISSTA 2019), pages 158–168. – URL https://doi.org/10.1145/3293882.3330580. – ISBN 9781450362245. DOI: 10.1145/3293882.3330580

[Cretin et al. 2020] CRETIN, A. ; VERNOTTE, A. ; CHEVROT, A. ; PEUREUX, F. ; LEGEARD, B.: **"Test Data Generation for False Data Injection Attack Testing in Air Traffic Surveillance"**. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pages 143–152

[Cretin et al. 2018] CRETIN, Aymeric ; LEGEARD, Bruno ; PEUREUX, Fabien ; VERNOTTE, Alexandre: **"Increasing the Resilience of ATC Systems against False Data Injection Attacks Using DSL-Based Testing"**. In *8th International Conference on Research in Air Transportation (ICRAT 2018)*, jun 2018, pages 1–4

[Dargie and Poellabauer 2010] DARGIE, Waltenegus ; POELLABAUER, Christian: *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley Publishing, 2010. – ISBN 0470997656

[Deursen et al. 2000] DEURSEN, Arie ; KLINT, Paul ; VISSER, Joost: **"Domain-Specific Languages: An Annotated Bibliography"**. In *SIGPLAN Notices* 35 (2000), 01, pages 26–36

[Dutta et al. 2020] DUTTA, I. K. ; GHOSH, B. ; CARLSON, A. ; TOTARO, M. ; BAYOUMI, M.: **"Generative Adversarial Networks in Security: A Survey"**. In *2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2020, pages 0399–0405. DOI: 10.1109/UEMCON51285.2020.9298135

[Ekstedt et al. 2015] EKSTEDT, M. ; JOHNSON, P. ; LAGERSTRÖM, R. ; GORTON, D. ; NYDRÉN, J. ; SHAHZAD, K.: **"Securi CAD by Foreseeti: A CAD Tool for Enterprise Cyber Security Management"**. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, 2015, pages 152–155. DOI: 10.1109/EDOCW.2015.40

[Enderlin 2014] ENDERLIN, Ivan: *Génération automatique de tests unitaires avec Praspel, un langage de spécification pour PHP*, Université de Franche-Comté ED SPIM, PhD Thesis, 2014. – URL http://www.theses.fr/2014BESA2067. – Thèse de doctorat dirigée par Bouquet, FabriceDadeau, Frédéric et Giorgetti, Alain Informatique Besançon 2014

[Esmalifalak et al. 2017] ESMALIFALAK, M. ; LIU, L. ; NGUYEN, N. ; ZHENG, R. ; HAN, Z.: **"Detecting Stealthy False Data Injection Using Machine Learning in Smart Grid"**. In *IEEE Systems Journal* 11 (2017), number 3, pages 1644–1652. DOI: 10.1109/JSYST.2014.2341597

[Farooq et al. 2015] FAROOQ, M. U. ; WASEEM, Muhammad ; KHAIRI, Anjum ; MAZHAR, Sadia: **"A Critical Analysis on the Security Concerns of Internet of Things (IoT)"**. In *International Journal of Computer Applications* (2015). DOI: 10.5120/19547-1280

[Felderer et al. 2016] FELDERER, Michael ; ZECH, Philipp ; BREU, Ruth ; BÜCHLER, Matthias ; PRETSCHNER, Alexander: **"Model-based security testing: a taxonomy and systematic classification: MODEL-BASED SECURITY TESTING"**. 26 (2016), number 2, pages 119–148. – URL http://doi.wiley.com/10.1002/stvr.1580. – Access date:: 2018-03-26. – ISSN 09600833. DOI: 10.1002/stvr.1580

[Frakes et al. 1998] FRAKES, W. ; PRIETO-DÍAZ, R. ; FOX, Christopher J.: **"DARE: Domain analysis and reuse environment"**. In *Annals of Software Engineering* 5 (1998), pages 125–141

[Fremantle and Scott 2017] FREMANTLE, Paul ; SCOTT, Philip: *A survey of secure middleware for the Internet of Things*. 2017. – URL https://peerj.com/articles/cs-114/. – Access date:: 2020-07-23

[Fremont et al. 2019] FREMONT, Daniel J. ; DREOSSI, Tommaso ; GHOSH, Shromona ; YUE, Xiangyu ; SANGIOVANNI-VINCENTELLI, Alberto L. ; SESHIA, Sanjit A.: **"Scenic: A Language for Scenario Specification and Scene Generation"**. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA : Association for Computing Machinery, 2019 (PLDI 2019), pages 63–78. – URL https://doi.org/10.1145/3314221.3314633. – ISBN 9781450367127. DOI: 10.1145/3314221.3314633

[Godefroid 2007] GODEFROID, Patrice: **"Random Testing for Security: Blackbox vs. Whitebox Fuzzing"**. In *Proceedings of the 2nd International Workshop on Random Testing: Co-Located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*. New York, NY, USA : Association for Computing Machinery, 2007 (RT '07), pages 1. – URL https://doi.org/10.1145/1292414.1292416. – ISBN 9781595938817. DOI: 10.1145/1292414.1292416

[Goodfellow et al. 2014] GOODFELLOW, Ian J. ; POUGET-ABADIE, Jean ; MIRZA, Mehdi ; XU, Bing ; WARDE-FARLEY, David ; OZAIR, Sherjil ; COURVILLE, Aaron ; BENGIO, Yoshua: **"Generative Adversarial Nets"**. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA : MIT Press, 2014 (NIPS'14), pages 2672–2680

[Hamdan et al. 2012] HAMDAN, Dima ; AKTOUF, Oum-El-Kheir ; PARISSIS, Ioannis ; ELHASSAN, Bachar ; HIJAZI, Abbas: **"Integrated Fault Tolerance Framework for Wireless Sensor Networks"**. In *2012 19th International Conference on Telecommunications, ICT 2012* (2012), 04. DOI: 10.1109/ICTEL.2012.6221282

[Hassija et al. 2019] HASSIJA, Vikas ; CHAMOLA, Vinay ; SAXENA, Vikas ; JAIN, Divyansh ; GOYAL, Pranav ; SIKDAR, Biplab: **"A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures"**. 7 (2019), pages 82721–82743. – URL https://ieeexplore.ieee.org/document/8742551/. – Access date:: 2020-07-24. – ISSN 2169-3536. DOI: 10.1109/ACCESS.2019.2924045

[Hoag and Thompson 2007] HOAG, Joseph E. ; THOMPSON, Craig W.: **"A Parallel General-Purpose Synthetic Data Generator"**. In *SIGMOD Rec.* 36 (2007), March, number 1, pages 19–24. – URL https://doi.org/10.1145/1276301.1276305. – ISSN 0163-5808. DOI: 10.1145/1276301.1276305

[Hulanicki et al. 1991] HULANICKI, A. ; GLAB, S. ; INGMAN, F.: **"Chemical sensors: definitions and classification"**. In *Pure and Applied Chemistry* 63 (1991), number 9, pages 1247 – 1250. – URL https://www.degruyter.com/view/journals/pac/63/9/article-p1247.xml

[ICS-CERT 2016] ICS-CERT: *Cyber-Attack Against Ukrainian Critical Infrastructure | CISA*. 2016. – URL https://us-cert.cisa.gov/ics/alerts/IR-ALERT-H-16-056-01. – Access date:: 2020-07-31

[ISO/IEC 1996] ISO/IEC: *ISO/IEC 14977:1996 Extended BNF*. 1996. – URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/61/26153.html. – Access date:: 2020-05-05. – Library Catalog: www.iso.org

[ITU 2012] ITU: *Overview of the Internet of Things - ITU-T Y.4000/Y.2060*. 2012

[Kang et al. 1990] KANG, Kyo ; COHEN, Sholom ; HESS, James ; NOVAK, William ; PETERSON, A.: *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. 1990. – URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231

[Khan et al. 2012] KHAN, Rafiullah ; KHAN, Sarmad U. ; ZAHEER, Rifaqat ; KHAN, Shahid: **"Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges"**. In *2012 10th International Conference on Frontiers of Information Technology*. Islamabad, Pakistan : IEEE, December 2012, pages 257–260. – ISBN 978-0-7695-4927-9 978-1-4673-4946-8. DOI: 10.1109/FIT.2012.53

[Kosar et al. 2016] KOSAR, Tomaž ; BOHRA, Sudev ; MERNIK, Marjan: **"Domain-Specific Languages: A Systematic Mapping Study"**. In *Information and Software Technology* 71 (2016), March, pages 77–91. – URL https://linkinghub.elsevier.com/

retrieve/pii/S0950584915001858. – Access date:: 2020-10-14. – ISSN 09505849. DOI: 10.1016/j.infsof.2015.11.001

[Li et al. 2019] LI, Dan ; CHEN, Dacheng ; JIN, Baihong ; SHI, Lei ; GOH, Jonathan ; NG, See-Kiong: **"MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks"**. In TETKO, Igor V. (editors) ; KŮRKOVÁ, Věra (editors) ; KARPOV, Pavel (editors) ; THEIS, Fabian (editors): *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*. Cham : Springer International Publishing, 2019, pages 703–716. – ISBN 978-3-030-30490-4

[Liang et al. 2017a] LIANG, Gaoqi ; WELLER, Steven R. ; ZHAO, Junhua ; LUO, Fengji ; DONG, Zhao Y.: **"The 2015 Ukraine Blackout: Implications for False Data Injection Attacks"**. In *IEEE Transactions on Power Systems* 32 (2017), July, number 4, pages 3317–3318. – ISSN 0885-8950, 1558-0679. DOI: 10.1109/TPWRS.2016.2631891

[Liang et al. 2017b] LIANG, Gaoqi ; ZHAO, Junhua ; LUO, Fengji ; WELLER, Steven R. ; DONG, Zhao Y.: **"A Review of False Data Injection Attacks Against Modern Power Systems"**. 8 (2017), number 4, pages 1630–1638. – URL http://ieeexplore.ieee.org/document/7438916/. – Access date:: 2020-07-29. – ISSN 1949-3053, 1949-3061. DOI: 10.1109/TSG.2015.2495133

[Liu et al. 2009] LIU, Yao ; NING, Peng ; REITER, Michael K.: **"False Data Injection Attacks against State Estimation in Electric Power Grids"**. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (2009), pages 16

[Lundin et al. 2002] LUNDIN, Emilie ; KVARNSTRÖM, Hkan ; JONSSON, Erland: **"A Synthetic Fraud Data Generation Methodology"**. In DENG, Robert (editors) ; BAO, Feng (editors) ; ZHOU, Jianying (editors) ; QING, Sihan (editors): *Information and Communications Security* Volume 2513. Springer Berlin Heidelberg, 2002, pages 265–277. – URL http://link.springer.com/10.1007/3-540-36159-6_23. – Access date:: 2020-09-02. – Series Title: Lecture Notes in Computer Science. – ISBN 978-3-540-00164-5 978-3-540-36159-6. DOI: 10.1007/3-540-36159-6_23

[Ma 2008] MA, Miao: **"Resilience Against False Data Injection Attack in Wireless Sensor Networks."**. In *Handbook of Research on Wireless Security*. IGI Global, 2008, pages 628–635. DOI: 10.4018/978-1-59904-899-4.ch038

[Mercuri and Neumann 2003] MERCURI, Rebecca T. ; NEUMANN, Peter G.: **"Security by obscurity"**. 46 (2003), number 11, pages 160. – URL http://portal.acm.org/citation.cfm?doid=948383.948413. – Access date:: 2020-05-19. – ISSN 00010782. DOI: 10.1145/948383.948413

[Mernik et al. 2005] MERNIK, Marjan ; HEERING, Jan ; SLOANE, Anthony M.: **"When and How to Develop Domain-Specific Languages"**. In *ACM Computing Surveys (CSUR)* 37 (2005), December, number 4, pages 316–344. – ISSN 0360-0300, 1557-7341. DOI: 10.1145/1118890.1118892

[Miao Wu et al. 2010] MIAO WU ; TING-JIE LU ; FEI-YANG LING ; JING SUN ; HUI-YING DU: **"Research on the Architecture of Internet of Things"**. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. Chengdu, China : IEEE, August 2010, pages V5–484–V5–487. – ISBN 978-1-4244-6539-2. DOI: 10.1109/ICACTE.2010.5579493

[Mo and Sinopoli 2010] MO, Yilin ; SINOPOLI, Bruno: **"False Data Injection Attacks in Control Systems"**. In *First Workshop on Secure Control Systems*, 2010, pages 7

[Mode et al. 2019] MODE, Gautam R. ; CALYAM, Prasad ; HOQUE, Khaza A.: **"False Data Injection Attacks in Internet of Things and Deep Learning enabled Predictive Analytics"**. (2019). – URL http://arxiv.org/abs/1910.01716. – Access date:: 2020-08-05

[Mokhtari et al. 2021] MOKHTARI, Sohrab ; ABBASPOUR, Alireza ; YEN, Kang K. ; SARGOLZAEI, Arman: **"A Machine Learning Approach for Anomaly Detection in Industrial Control Systems Based on Measurement Data"**. In *Electronics* 10 (2021), number 4. – URL https://www.mdpi.com/2079-9292/10/4/407. – ISSN 2079-9292. DOI: 10.3390/electronics10040407

[NETSCOUT 2018] NETSCOUT: *NETSCOUT Threat Intelligence Report - Second Half 2018*. 2018

[Ngu et al. 2016] NGU, Anne H. H. ; GUTIERREZ, Mario ; METSIS, Vangelis ; NEPAL, Surya ; SHENG, Michael Z.: **"IoT Middleware: A Survey on Issues and Enabling technologies"**. (2016), pages 1–1. – URL http://ieeexplore.ieee.org/document/7582463/. – Access date:: 2020-07-23. – ISSN 2327-4662. DOI: 10.1109/JIOT.2016.2615180

[Padmavathi and Shanmugapriya 2009] PADMAVATHI, Dr G. ; SHANMUGAPRIYA, Mrs D.: **"A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks"**. In *International Journal of Computer Science and Information Security, IJCSIS, Vol. 4, No. 1 & 2, August 2009, USA* 4 (2009), number 1, pages 10

[Popić et al. 2019] POPIĆ, S. ; PAVKOVIĆ, B. ; VELIKIĆ, I. ; TESLIĆ, N.: **"Data generators: a short survey of techniques and use cases with focus on testing"**. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, 2019, pages 189–194. DOI: 10.1109/ICCE-Berlin47944.2019.8966202

[Qingyu Yang et al. 2014] QINGYU YANG ; JIE YANG ; WEI YU ; DOU AN ; NAN ZHANG ; WEI ZHAO: **"On False Data-Injection Attacks against Power System State Estimation: Modeling and Countermeasures"**. 25 (2014), number 3, pages 717–729. – URL http://ieeexplore.ieee.org/document/6490324/. – Access date:: 2020-04-27. – ISSN 1045-9219. DOI: 10.1109/TPDS.2013.92

[Rabl and Poess 2011] RABL, Tilmann ; POESS, Meikel: **"Parallel Data Generation for Performance Analysis of Large, Complex RDBMS"**. In *Proceedings of the Fourth International Workshop on Testing Database Systems*. New York, NY, USA : Association for Computing Machinery, 2011 (DBTest '11), pages 6. – URL https://doi.org/10.1145/1988842.1988847. – ISBN 9781450306553. DOI: 10.1145/1988842.1988847

[Santos et al. 2017] SANTOS, Joanna C. S. ; TARRIT, Katy ; MIRAKHORLI, Mehdi: **"A Catalog of Security Architecture Weaknesses"**. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. Gothenburg, Sweden : IEEE, April 2017, pages 220–223. – ISBN 978-1-5090-4793-2. DOI: 10.1109/ICSAW.2017.25

[Schlegel et al. 2011] SCHLEGEL, Roman ; ZHANG, Kehuan ; ZHOU, Xiaoyong ; INTWALA, Mehool ; KAPADIA, Apu ; WANG, XiaoFeng: **"Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones"**. (2011), pages 17

[Sencun Zhu et al. 2004] SENCUN ZHU ; SETIA, S. ; JAJODIA, S. ; PENG NING: **"An Interleaved Hop-by-Hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks"**. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. Berkeley, CA, USA : IEEE, 2004, pages 259–271. – ISBN 978-0-7695-2136-7. DOI: 10.1109/SECPRI.2004.1301328

[Sikder et al. 2018] SIKDER, Amit K. ; PETRACCA, Giuseppe ; AKSU, Hidayet ; JAEGER, Trent ; ULUAGAC, A. S.: **"A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications"**. In *CoRR* abs/1802.02041 (2018). – URL http://arxiv.org/abs/1802.02041

[Swamy et al. 2017] SWAMY, Sowmya N. ; JADHAV, Dipti ; KULKARNI, Nikita: **"Security Threats in the Application layer in IOT Applications"**. (2017), pages 4

[Tairas et al. 2009] TAIRAS, Robert ; MERNIK, Marjan ; GRAY, Jeff: **"Using Ontologies in the Domain Analysis of Domain-Specific Languages"**. In CHAUDRON, Michel R. V. (editors): *Models in Software Engineering*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, pages 332–342. – ISBN 978-3-642-01648-6

[Uluagac et al. 2014] ULUAGAC, A. S. ; SUBRAMANIAN, Venkatachalam ; BEYAH, Raheem: **"Sensory Channel Threats to Cyber Physical Systems: A Wake-up Call"**. In *2014 IEEE Conference on Communications and Network Security*. San Francisco,

CA, USA : IEEE, October 2014, pages 301–309. – ISBN 978-1-4799-5890-0. DOI: 10.1109/CNS.2014.6997498

[Utting and Legeard 2007] UTTING, Mark ; LEGEARD, Bruno: *Practical model-based testing: a tools approach*. Morgan Kaufmann Publishers, 2007. – OCLC: ocm73993672. – ISBN 978-0-12-372501-1

[Utting et al. 2016] UTTING, Mark ; LEGEARD, Bruno ; BOUQUET, Fabrice ; FOURNERET, Elizabeta ; PEUREUX, Fabien ; VERNOTTE, Alexandre: **"Recent Advances in Model-Based Testing"**. In *Advances in Computers* Volume 101. Elsevier, 2016, pages 53–120. – URL https://linkinghub.elsevier.com/retrieve/pii/S0065245815000650. – Access date:: 2020-08-13. – ISBN 978-0-12-805158-0. DOI: 10.1016/bs.adcom.2015.11.004

[Utting et al. 2012] UTTING, Mark ; PRETSCHNER, Alexander ; LEGEARD, Bruno: **"A taxonomy of model-based testing approaches"**. 22 (2012), number 5, pages 297–312. – URL http://doi.wiley.com/10.1002/stvr.456. – Access date:: 2020-08-11. – ISSN 09600833. DOI: 10.1002/stvr.456

[White 1987] WHITE, R.M.: **"A Sensor Classification Scheme"**. In *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control* 34 (1987), March, number 2, pages 124–126. – ISSN 0885-3010. DOI: 10.1109/T-UFFC.1987.26922

[Xie et al. 2010] XIE, Le ; MO, Yilin ; SINOPOLI, Bruno: **"False Data Injection Attacks in Electricity Markets"**. (2010), pages 6

[Yang et al. 2017] YANG, Lijun ; DING, Chao ; WU, Meng ; WANG, Kun: **"Robust Detection of False Data Injection Attacks for Data Aggregation in an Internet of Things-Based Environmental Surveillance"**. In *Computer Networks* 129 (2017), December, pages 410–428. – ISSN 13891286. DOI: 10.1016/j.comnet.2017.05.027

[Ye and Zhang 2020] YE, Dan ; ZHANG, Tian-Yu: **"Summation Detector for False Data-Injection Attack in Cyber-Physical Systems"**. 50 (2020), number 6, pages 2338–2345. – URL https://ieeexplore.ieee.org/document/8727926/. – Access date:: 2020-08-03. – ISSN 2168-2267, 2168-2275. DOI: 10.1109/TCYB.2019.2915124

[Yi Huang et al. 2011] YI HUANG ; LI, Husheng ; CAMPBELL, Kristy A. ; ZHU HAN: **"Defending false data injection attack on smart grid network using adaptive CUSUM test"**. In *2011 45th Annual Conference on Information Sciences and Systems*, IEEE, 2011, pages 1–6. – URL http://ieeexplore.ieee.org/document/5766111/. – Access date:: 2020-08-10. – ISBN 978-1-4244-9846-8. DOI: 10.1109/CISS.2011.5766111

[Zhao and Ge 2013] ZHAO, Kai ; GE, Lina: **"A Survey on the Internet of Things Security"**. In *2013 Ninth International Conference on Computational Intelligence and Security*. Emeishan 614201, China : IEEE, December 2013, pages 663–667. – ISBN 978-1-4799-2549-0 978-1-4799-2548-3. DOI: 10.1109/CIS.2013.145

List of Figures

2.1 Simplified infrastructure of the parking management system 14
2.2 2021 typical parking meter 15
2.3 securiCAD modelling of one IPS services for risk analysis purposes 18
2.4 Analogy between the architecture of an IoT system and an IPS 21

3.1 Query 1: Publication where the title contains terms covering FDIA 35
3.2 Query 2: Query 1 + containing the terms covering IoT 35

4.1 Categorisation of a data 50

5.1 Example of a FDIA scenario 60
5.2 EBNF Grammar of a scenario 60
5.3 EBNF Grammar of the scenario actions 61
5.4 EBNF Grammar of the selection criteria 62
5.5 EBNF Grammar of the alteration criteria 63
5.6 EBNF Grammar of the time frame 64
5.7 Grammar of utility elements 65
5.8 Complete grammar of the FD2IOT language 66
5.9 UML class diagram of the implementation in java of the interpretation 68

6.1 Workflow for FDIA testing - Data acquisition, designing and testing 70
6.2 Example of a JSON flattenning 72
6.3 Scenario example of a test 73
6.4 Scenario example of a test 74
6.5 Scenario example of a test 74
6.6 Alteration primitive level 75
6.7 Hypothetical scenario for a specific alteration 76

6.8 Hypothetical Scenario for a specific alteration 79

7.1 Data gathered by a thing in JSON 84
7.2 FDIA on temperature in degree Celsius in May 2019 85
7.3 FDIA for sensor disruption 85
7.4 Map of the sensor situation 86
7.5 Particles in may 2019 before an FDI attack 86
7.6 FDIA for particle increase with distance attenuation 87
7.7 Particles in may 2019 after the FDIA execution 87
7.8 Average noise in decibels 94
7.9 lightGBM training 94
7.10 lightGBM training zoomed 95
7.11 Detection with lightGBM on a random value attack 95
7.12 Scenario for stealth noise attack 96
7.13 Distribution of average decibels by noise categories before and after FDIA . 96
7.14 Detection of the attack on lowest and highest value of the noise histogram . 97
7.15 Average noise during the month of March 2020 98

List of Tables

3.1 Architecture layers and their security issues 26
3.2 Summary of FDIA definition 34
3.3 Query on Google Scholar of different domains associated with FDIA 36
4.1 Sensor classification 48
7.1 Overview of efficiency experiments 91

List of Definitions

1 Definition: Internet of Things 5
2 Definition: Thing 5
3 Definition: False Data Injection Attack 34
4 Definition: Domain-Specific Language 54

ACRONYMS

- **ANPR:** Automatic Number-Plate Recognition. 17
- **AST:** Abstract Syntax Tree. 66, 67
- **ATC:** Air Traffic Control. 10, 36, 41, 76
- **CPS:** Cyber Physical Systems. 32, 34, 36, 43, 106
- **DAST:** Dynamic Application Security Testing. 38
- **DoS:** Denial of Service. 19, 27, 28, 30
- **DSL:** Domain Specific Language. 6, 7, 9, 12, 36, 41, 42, 43, 44, 53, 54, 55, 56, 57, 59, 60, 64, 65, 67, 68, 69, 70, 71, 73, 74, 76, 77, 79, 84, 87, 92, 93, 102, 103, 104, 105
- **EBNF:** Extended Backus-Naur Form. 60, 64
- **FDI:** False Data Injection. 31
- **FDIA:** False Data Injection Attack. 4, 6, 7, 8, 9, 10, 11, 12, 17, 23, 25, 27, 28, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 47, 49, 50, 51, 52, 55, 56, 57, 58, 59, 60, 61, 67, 68, 69, 70, 71, 74, 75, 76, 77, 78, 79, 83, 84, 85, 87, 89, 90, 92, 93, 95, 96, 97, 98, 99, 101, 102, 103, 105, 106
- **GAN:** Generative Adversarial Networks. 105, 106
- **GeLeaD:** Generate & Learn & Detect. 10
- **GPL:** General Programming Language. 54, 56, 65
- **IoT:** Internet of Things. 4, 5, 6, 7, 8, 9, 10, 11, 14, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 33, 34, 35, 36, 41, 42, 43, 44, 47, 50, 51, 52, 55, 57, 58, 60, 68, 69, 70, 71, 72, 75, 77, 78, 79, 84, 90, 92, 101, 102, 103, 104, 105
- **IPS:** Intelligent Parking Systems. 10, 48, 49, 88, 92, 98
- **ITU:** International Telecommunication Union. 5

- **KDT:** Keyword Driven Testing. 105
- **MBFST:** Model-Based Functional Security Testing. 38
- **MBST:** Model-Based Security Testing. 37, 38
- **MBT:** Model-Based Testing. 6, 37, 38, 43, 105
- **MBVST:** Model-Based Vulnerability Security Testing. 38
- **MITM:** Man-In-The-Middle. 19, 92
- **ML:** Machine Learning. 11, 83, 93, 96, 98, 105
- **OS:** Operating System. 15
- **PA DSS:** Payment Application Data Security Standard. 18
- **PCI DSS:** Payment Card Industry Data Security Standard. 16, 18
- **SARCoS:** Secure And Reliable Connected Systems. 10
- **SUT:** System Under Test. 37, 38, 60, 63, 64, 70, 71, 72, 78, 92, 104
- **WSN:** Wireless Sensor Network. 30, 31, 32, 34, 35, 36, 43

IV

APPENDIX

A

ANTLR GRAMMAR DEFINITION

```
grammar fd2iot;

SCENARIO: 'scenario';
TICKER:'ticker';
CREATE:'create';
ALTER:'alter';
COPY:'copy';
DELETE:'delete';
THINGS:'things';
SET:'set';
FROM:'from';
TO:'to';
OF:'of';
ISINSIDECIRCLE:'isInsideCircle';
WITH: 'with';
ATTENUATION: 'attenuation';
GEOLOCATION: 'geolocation';
ARRAY: 'array';
EQUAL_SYMBOL:'=';
GREATER_SYMBOL:'>';
LESS_SYMBOL:'<';
DIFFERENT_SYMBOL:'!=';
EXCLAMATION_SYMBOL:'!';
INCREMENT_SYMBOL:'+=';
MULTIPLYOFFSET_SYMBOL:'*=';
WHERE:'where';
SEMI: ';';
COMMA: ',';
AND:'and';
LEFT_BRACKET:'(';
RIGHT_BRACKET:')';
LEFT_ARROW:'->';
```

```
STRING_LITERAL: DQUOTA_STRING | SQUOTA_STRING | BQUOTA_STRING;
DECIMAL_LITERAL: ('-')? DEC_DIGIT+;
HEXADECIMAL_LITERAL: 'X' '\'' (HEX_DIGIT HEX_DIGIT)+ '\''
                   | '0X' HEX_DIGIT+;
ID:  ID_LITERAL;

REAL_LITERAL:                       ('-')?((DEC_DIGIT+)? '.' DEC_DIGIT+
                                    | DEC_DIGIT+ '.' EXPONENT_NUM_PART
                                    | (DEC_DIGIT+)? '.' (DEC_DIGIT+ EXPONENT_NUM_PART)
                                    | DEC_DIGIT+ EXPONENT_NUM_PART);

fragment ID_LITERAL:                [A-Za-z_$*]*?[A-Za-z_$*]+?[A-Za-z_$*0-9]*;
fragment DQUOTA_STRING:             '"' ( '\\'. | '""' | ~('"'| '\\') )* '"';
fragment SQUOTA_STRING:             '\'' ('\\'. | '\'\'' | ~('\'' | '\\'))* '\'';
fragment BQUOTA_STRING:             '`' ( '\\'. | '``' | ~('`'|'\\'))* '`';
fragment HEX_DIGIT:                 [0-9A-F];
fragment DEC_DIGIT:                 [0-9];
fragment BIT_STRING_L:              'B' '\'' [01]+ '\'';
fragment EXPONENT_NUM_PART:         'E' [-+]? DEC_DIGIT+;

WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newline


root: scenarioDeclaration executionList;

scenarioDeclaration:
        SCENARIO STRING_LITERAL TICKER DECIMAL_LITERAL
            (GEOLOCATION LEFT_BRACKET REAL_LITERAL COMMA REAL_LITERAL RIGHT_BRACKET)
;

executionList:
        execution SEMI (execution SEMI)*
;

execution: ((CREATE THINGS alterationCriteria timeframe)
        |  (ALTER THINGS selectionCriteria alterationCriteria timeframe)
        | (DELETE THINGS selectionCriteria timeframe)
        | (COPY THINGS selectionCriteria alterationCriteria timeframe))
        ;

selectionCriteria: WHERE selectionCriterion (AND selectionCriterion)*;

selectionCriterion: ID EQUAL_SYMBOL type #equalSelection
        | ID GREATER_SYMBOL type #greaterThanSelection
```

```
        | ID LESS_SYMBOL type  #lessThanSelection
        | ID DIFFERENT_SYMBOL type #differentSelection
        | ID ISINSIDECIRCLE LEFT_BRACKET REAL_LITERAL COMMA REAL_LITERAL
            COMMA DECIMAL_LITERAL RIGHT_BRACKET #isInsideCircleSelection
        ;


timeframe: FROM DECIMAL_LITERAL TO DECIMAL_LITERAL;

attenuationCriteria: WITH ATTENUATION OF REAL_LITERAL;

alterationCriteria: SET alterationCriterion (AND alterationCriterion)*;

alterationCriterion: ID EQUAL_SYMBOL type #affectation
|ID EQUAL_SYMBOL evol #evolution
|ID INCREMENT_SYMBOL REAL_LITERAL #offset
|ID MULTIPLYOFFSET_SYMBOL REAL_LITERAL #multiplyOffset
|ID INCREMENT_SYMBOL evol attenuationCriteria? #evolutionIncrementation
|array INCREMENT_SYMBOL REAL_LITERAL #arrayIncrementation
;

evol: LEFT_BRACKET REAL_LITERAL LEFT_ARROW REAL_LITERAL COMMA
    REAL_LITERAL RIGHT_BRACKET;

array: ARRAY LEFT_BRACKET ID COMMA DECIMAL_LITERAL COMMA
    DECIMAL_LITERAL RIGHT_BRACKET;

type:ID #identType
|STRING_LITERAL #stringLiteralType
|DECIMAL_LITERAL #decimalLitealType
|REAL_LITERAL #realLiteralType
|HEXADECIMAL_LITERAL #hexaDecimalType;
```

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — https://github.com/gallandarakhneorg/tex-templates
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

école doctorale **sciences pour l'ingénieur et microtechniques**

Title: Addressing False Data Injection Attacks in IoT Systems with a Domain Specific Language within an Industrial Context

Keywords: Internet of Things, False Data Injection Attack, Test, Security, Domain-Specific Language

Abstract:

The Internet of Things (IoT) is nowadays ubiquitous in all spheres of life. Its applications can be found in many fields, such as smart cities, industry, health, home automation, etc. The rapid explosion of the Internet of Things has seen the emergence of numerous security issues, notably due to the great heterogeneity of "things" and their protocols, as well as the lack of dedicated standards. This thesis focuses on a specific security problem existing within the IoT, False Data Injection Attacks (FDIA). Indeed, the services provided by the IoT rely on the use of data retrieved by numerous sensors. One of the many attack vectors is then the data transiting within the devices. To allow the description and the implementation of these attacks, we propose an approach using a Domain-Specific Language allowing to model FDIA, to test the resilience of IoT systems and to help the development of detection tools for such attacks. This work took place within the company Flowbird, the world leader in parking solutions, which allowed us to use its devices to develop our approach.

Titre : Adresser les attaques par injection de fausses données dans les systèmes IoT à l'aide d'un langage dédié dans un contexte industriel

Mots-clés : Internet des Objets, Attaques par injection de données altérés, Test, Securité, Langage dédié

Résumé :

L'internet des objets (IoT) est aujourd'hui omniprésent dans toutes les sphères de la vie. Ses applications se trouvent dans de nombreux domaines, tels que les villes intelligentes, l'industrie, la santé, domotique, etc. L'explosion rapide de l'internet des objets a vu l'émergence de nombreuses problématiques de sécurité, notamment à cause de la grande hétérogénéité des objets connectés et de leurs protocoles, ainsi que du manque de standard dédié. Cette thèse se concentre sur une problématique de sécurité existant au sein de l'IoT : les attaques par injection de données altérées. En effet, les services fournis par l'IoT reposent sur l'utilisation des données récupérées par de nombreux capteurs. L'un des vecteurs d'attaques est alors de modifier en amont les données transitant au sein des dispositifs et ainsi duper celui-ci ou ses utilisateurs. Pour permettre la description puis la mise en oeuvre de ces attaques, nous proposons une approche utilisant un langage dédié à cette modélisation permettant de spécifier les attaques par injection de données altérées, de tester la résilience des systèmes IoT et d'aider aux développements des outils de détection de telles attaques. Ces travaux se sont déroulés au sein de l'entreprise Flowbird, leader mondial de solutions de stationnements, qui nous a permis d'utiliser ses dispositifs pour développer notre approche.

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon, France